LTP 服务接口调用文档

韩中华
zhhan@ir.hit.edu.cn
车万翔，李正华指导

2010-1-3

目录#

LTML 规范

LTP 对外提供服务的方式是通过 HTTP 协议进行的。客户端请求（文本或 XML 串），服务器端返回相应的分析结果（XML 串），返回结果如下图所示，服务器的默认编码方式为 gbk。

```
<?xml version="1.0" encoding=" gbk " ?>
<xml4nlp>
    <note sent="y" word="y" pos="y" ne="y" parser="y" wsd="y" srl="y" />
    <doc>
        <para id="0">
            <sent id="0" cont="我们都是中国人">
                <word id="0" cont="我们" wsd="Aa02" wsdexp="我_我们" pos="r" ne="O" parent="2" relate="SBV" />
                <word id="1" cont="都" wsd="Ka07" wsdexp="都_只_不止_甚至" pos="d" ne="O" parent="2" relate="ADV" />
                <word id="2" cont="是" wsd="Ja01" wsdexp="是_当做_比作" pos="v" ne="O" parent="-1" relate="HED">
                    <arg id="0" type="A0" beg="0" end="0" />
                    <arg id="1" type="AM-ADV" beg="1" end="1" />
                </word>
                <word id="3" cont="中国" wsd="Di02" wsdexp="国家_行政区划" pos="ns" ne="S-Ns" parent="4" relate="ATT" />
                <word id="4" cont="人" wsd="Aa01" wsdexp="人_人民_众人" pos="n" ne="O" parent="2" relate="VOB" />
            </sent>
        </para>
    </doc>
</xml4nlp>
```

LTP 数据表示标准称为 LTML。LTML 标准要求如下；

结点标签分别为 xml4nlp, note, doc, para, sent, word, arg 共七种结点标签；

1. xml4nlp 为根结点，无任何属性值；
2. note 为标记结点，具有的属性分别为：sent, word, pos, ne, parser, wsd, srl；分别代表分句，分词，词性标注，命名实体识别，依存句法分析，词义消歧，语义角色标注；值为”n”，表明未做，值为”y”则表示完成，如 pos=”y”，表示已经完成了词性标注；
3. doc 为篇章结点，以段落为单位包含文本内容；无任何属性值；
4. para 为段落结点，需含 id 属性，其值从 0 开始；
5. sent 为句子结点，需含属性为 id，cont；id 为段落中句子序号，其值从 0 开始；cont 为句子内容；
6. word 为分词结点，需含属性为 id, cont；id 为句子中的词的序号，其值从 0 开始，cont 为分词内容；

   可选属性为 pos, ne, wsd, wsdexp, parent, relate；pos 的内容为词性标注内容；ne 为命名实体内容；wsd 与 wsdexp 成对出现，wsd 为词义消歧内容，wsdexp 为相应的解释说明；parent 与 relate 成对出现，parent 为依存句法分析的父亲结点 id 号，relate 为相对应的关系；
7. arg 为语义角色信息结点，任何一个谓词都会带有若干个该结点；其属性为 id, type, beg, end；id 为序号，从 0 开始；type 代表角色名称；beg 为开始的词序号，end 为结束的序号；

各结点及属性的逻辑关系说明如下：

1. 各结点层次关系可以从图中清楚获得，凡带有 id 属性的结点是可以包含多个；
2. 如果 sent=”n”即未完成分句，则不应包含 sent 及其下结点；
3. 如果 sent=”y” word=”n”即完成分句，未完成分词，则不应包含 word 及其下结点；
4. 其它情况均是在 sent=”y” word=”y”的情况下：
   - ✓ 如果 pos=”y”则分词结点中必须包含 pos 属性；
   - ✓ 如果 ne=”y”则分词结点中必须包含 ne 属性；
   - ✓ 如果 parser=”y”则分词结点中必须包含 parent 及 relate 属性；
   - ✓ 如果 wsd=”y”则分词结点中必须包含 wsd 及 wsdexp 属性；
   - ✓ 如果 srl=”y”则凡是谓词（predicate）的分词会包含若干个 arg 结点；

LTP 标注集及说明

词性标注集及含义

LTP 使用的是 863 词性标注集，其各个词性含义如下表。

Tag	Description	Example	Tag	Description	Example
a	adjective	美丽	ni	organization name	保险公司
b	other noun-modifier	大型，西式	nl	location noun	城郊
c	conjunction	和，虽然	ns	geographical name	北京
d	adverb	很	nt	temporal noun	近日，明代
e	exclamation	哎	nz	other proper noun	诺贝尔奖
g	morpheme	茨，甥	o	onomatopoeia	哗啦
h	prefix	阿，伪	p	preposition	在，把
i	idiom	百花齐放	q	quantity	个
j	abbreviation	公检法	r	pronoun	我们
k	suffix	界，率	u	auxiliary	的，地
m	number	一，第一	v	verb	跑，学习
n	general noun	苹果	wp	punctuation	，。！
nd	direction noun	右侧	ws	foreign words	CPU
nh	person name	杜甫，汤姆	x	non-lexeme	萄，翱

命名实体标注体系及含义

NE 识别模块的标注结果采用 O-S-B-I-E 标注形式，具体你可以参考一些 NE 标注的文章。 其含义是：

O 表示这个词不是 NE

S 表示这个词单独构成一个 NE

B 表示这个词为一个 NE 的开始

I 表示这个词为一个 NE 的中间

E 表示这个词位一个 NE 的结尾

我们实验室的 NE 模块识别七种 NE，分别如下：

Nm	数词
Ni	机构名
Ns	机构名
Nh	人名
Nt	时间
Nr	日期
Nz	专有名词

示例：

巴格达	南部	地区
B-Ns	I-Ns	E-Ns

说明这三个词构成一个 NE（地名）。

20 日	上午
B-Nr	E-Nr

说明这两个词构成一个 NE（日期）。

依存句法标注体系及含义

关系	符号	关系	符号
定中关系	ATT(attribute)	“的”字结构	DE
数量关系	QUN(quantity)	“地”字结构	DI
并列关系	COO(coordinate)	“得”字结构	DEI
同位关系	APP(appositive)	“把”字结构	BA
前附加关系	LAD(left adjunct)	“被”字结构	BEI
后附加关系	RAD(right adjunct)	状中结构	ADV(adverbial)
比拟关系	SIM(similarity）	动宾关系	VOB(verb-object)
语态结构	MT(mood-tense)	主谓关系	SBV(subject-verb)
独立结构	IS(indep. structure)	连动结构	VV(verb-verb)
动补结构	CMP(complement）	关联结构	CNJ(conjunctive)
介宾关系	POB(prep-obj)	独立分句	IC(indep. clause)
核心	HED (head)	依存分句	DC(dep. clause)
分析器无法确定的关系	NOT		

具体可参考马金山博士的毕业论文，论文链接。

http://ir.hit.edu.cn/demo/ltp/SharingPackage/mjs-dissertation.pdf

LTP 服务接口

C++接口

LTP 需要与服务器进行交互，以完成对文本的分析，分析的返回结果存储在以 DOM 形式组织的 XML 中，客户端接到分析结果后可通过提供的接口对结果进行分析。

C++ 接口的操作可分为如下两个类
LTPService类为与服务器交互类，负责验证、连接分析参数设置等；
LTML类为返回数据(XML)解析函数，专门负责数据的生成和提取；
LTP的C++各类均定义在命名空间HIT_IR_LTP中。因此在书写程序时需包含：

using namespace HIT_IR_LTP;

跨平台编译说明 [WINDOWS] [LINUX]

无论[WINDOWS] 还是[LINUX]，首先必需修改头文件LTPOption.h（位于__ltpService目录下），注释掉系统LINUX_OS或WIN_OS；

[WINDOWS]

在LTPOption.h中，注释掉“#define LINUX_OS”,保留“#define WIN_OS”；

开发IDE为Microsoft Visual Studio 2008，值得注意的是项目中只能有一个main函数入口，在给的若干个例程中（Example）都包含有main函数；

[LINUX]

在LTPOption.h中，注释掉“#define WIN_OS”,保留“#define LINUX_OS”；

安装LINUX下所需的库，运行如下命令：

```
./configure
make
make install
```

编译例程，命令如下：

```
g++ example1.cpp -I /usr/ local/include \
   -L /usr/ local/lib -lutil -lxml4nlp –lservice \
   -o test
```

执行./test即可；

具体接口如下：

LTPService 类

LTPService.h 位于__ltpService 内；
该类主要负责与服务器交互，并将返回结果以 Ltml 对象返回。

- ✓ LTPService(const std::string& authorization)
  构造函数，authorization 为用户验证信息，信息格式为：**username:password**；

- ✓ bool SetEncoding(const std::string& encodingType)
  设置字符编码，默认为 gbk（LTPOption.GBK），目前仅支持 UTF-8(LTPOption.UTF8)及 GBK，GB2312。编码的定义请参考 LTPOption.h;

- ✓ bool Analyze(const std::string& option, const std::string& analyzeString, LTML& ltml_out)
  参数 option 为分析方式，分析的方式包括：分词（LTPOption.WS），词性标注（LTPOption.POS），命名实体识别（LTPOption.NE），词义消歧（LTPOption.WSD），依存句法分析（LTPOption. PARSER），语义角色标注（LTPOption.SRL）。
  参数 analyzeString 为待分析字符串（文本串，非 XML 串）。
  ltml_out 对象为返回的 LTML 类型（各种 set 方法是对分析属性的设置，应在本操作之前）；
  用户验证通过，分析成功返回 true，验证未通过，返回 false；

- ✓ bool Analyze(const std::string& option, const LTML& ltml_in, LTML& ltml_out)
  参数 option 为分析方式，分析的方式包括：分词（LTPOption.WS），词性标注（LTPOption.POS），命名实体识别（LTPOption.NE），词义消歧（LTPOption.WSD），依存句法分析（LTPOption. PARSER），语义角色标注（LTPOption.SRL）。
  参数 ltml_in 为待分析的 Ltml 对象，其内容可以由用户指定。
  ltml_out 对象为返回的 Ltml 类型（各种 set 方法是对分析属性的设置，应在本操作之前）；
  用户验证通过，分析成功返回 true，验证未通过，返回 false；

LTML 类

LTML.h 位于__ltpService 内；
LTML 类提供 XML 操作方法，包括 XML 的生成，XML 中信息的提取。
该类是对返回的数据(XML 串)进行解析的主要对象。

- ✓ int LoadLtml(const std::string & str)
  从 XML 串生成 LTML 对象

- ✓ bool HasSent ()
  返回结果是否进行了分句。是返回 true，否返回 false。

- ✓ bool HasWS ()

返回结果是否进行了分词。是返回 true，否返回 false。

- ✓ bool HasPOS ()
  返回结果是否进行了词性标注。是返回 true，否返回 false。

- ✓ bool HasNE ()
  返回结果是否进行了命名实体分析。是返回 true，否返回 false。

- ✓ bool HasParser ()
  返回结果是否进行了依存句法分析。是返回 true，否返回 false。

- ✓ bool HasWSD ()
  返回结果是否进行了词义消歧分析。是返回 true，否返回 false。

- ✓ bool HasSRL ()
  返回结果是否进行了语义角色标注。是返回 true，否返回 false；

（Count 段落数、句子数、词语数）

- ✓ int CountParagraph ()
  分析结果中段落数。

- ✓ int CountSentence ()
  分析结果中句子数。

- ✓ int CountSentence (int paragraphIdx)
  分析结果中，段落 paragraphIdx 中的句子数；
  参数 paragraphIdx 为段落号。

- ✓ bool GetWords(vector<Word> &wordList, int paragraphIdx, int sentenceIdx)
  返回分析结果中，第 paragraphIdx 段的第 sentenceIdx 句中的所有词，返回结果存储在 wordList 变量中，当成功时，返回 true，失败时返回 false。
  paragraphIdx 为段落号，sentenceIdx 相应段落中的句子号。

- ✓ bool GetWords (vector<Word> &wordList, int sentenceIdx)
  返回分析结果中，在整篇文章的第 sentenceIdx 名中的所有词，返回结果存储在 wordList 变量中，当成功时，返回 true，失败时返回 false。
  sentenceIdx 为整篇文章中句子的序号。

- ✓ bool GetSentenceContent(string &content, int paragraphIdx, int sentenceIdx)
  返回分析结果中，第 paragraphIdx 段的第 sentenceIdx 句的内容，返回结果存储在 content 变量中，当成功时，返回 true，失败时返回 false。
  paragraphIdx 为段落号，sentenceIdx 相应段落中的句子号。

- ✓ bool GetSentenceContent(string &content, int globalSentIdx)

返回分析结果中，在整篇文章的第 globalSentIdx 句的内容，返回结果存储在 content 变量中，当成功时，返回 true，失败时返回 false。
globalSentenceId 为整篇文章中句子的序号。

- ✓ void SaveDOM(const char* fileName)
  将 LTML 对象保存到 XML 文件中。

- ✓ std::string GetXMLStr()
  将 LTML 对象写到字符串中。

- ✓ void ClearDOM()
  将 LTML 对象清空；

以下几个方法是向 LTML 对象写数据的方法。注意以下几个问题：

1. 请保证调用以下方法的 LTML 对象为空的，或首先调用过 ClearDOM 方法，以保证 LTML 对象中数据一致；
2. 输入的数据必须保证一致，例如，如果第一个词完成了词性标注，则其余词也须完成词性标注，因为 LTML 对象是根据第一个词或第一句话生成的 note 结点；
3. 已经调用 SetOver 的 LTML 对象不允许调用以下方法，否则会抛异常；凡是由 LTPService 对象返回的 Ltml 对象都调用过 SetOver 方法；

- ✓ bool SetParagraphNumber(int paragraphNumber)
  设置段落数量。注意该方法只能向空的（初始）LTML 对象中设置，否则会抛出异常；如果不调用该方法，默认的段落数量为 1。

- ✓ void AddSentence(const vector<Word> &wordList, int paragraphId)
  向第 paragraphIdx 个段落中增加一个句子；
  参数 wordList 为句子中的词列表；note 结点是根据第一个 sentence 第一个词生成的，必须保证后面句子中的词与第一个词一致，否则会抛异常；
  参数 paragraphIdx 为插入的段落号（从 0 开始）；

- ✓ void AddSentence(const std::string sentenceContent, int paragraphId)
  向第 paragraphId 个段落中增加一个句子，该方法适用于只完成了分句的情况；
  sentenceContent 为句子内容。注意，如果已经增加分词的 Ltml 对象调用该方法，会抛出异常；

- ✓ void setOver()
  一旦调用该方法，LTML 对象就将不能再调用 AddSentence 及 SetParagraphNumber，每次由 Analyze 方法返回的 LTML 对象都会调用该方法；如，在调用 AddSentence 添加完所有的句子后，调用 setOver 表明调用结束；

Word 类型

Word.h 位于__ltpService 内；
该类型是对 XML 的 Element 进行的封装，任何的分析结果必须先取到 Word，才能取到相当相应的分析数据。

- ✓ Word()
  构造函数，构造一个空的 Word 对象。

- ✓ bool HasID ()
  是否有 ID 号，有返回 true，否则返回 false；此 ID 为由服务器端生成的该词在句子中的唯一 ID 号，因此，此 ID 只在句子中才有意义；

- ✓ int GetID ()
  返回词的 ID 号。此 ID 为由服务器端生成的该词在句子中的唯一 ID 号，因此，此 ID 只在句子中才有意义。成功是，返回值大于等于 0（>=0），失败时返回值为-1。

- ✓ void SetID (int id)
  设置词在句子中的 ID 号；

- ✓ bool HasWS ()
  是否进行了分词，是返回 true，否则返回 false；

- ✓ string GetWS ()
  返回词的具体内容。失败时返回空(NULL)。

- ✓ void SetWS (const std::string& content)
  设置分词内容；

- ✓ bool HasPOS ()
  是否进行了词性标注，是返回 true，否则返回 false；

- ✓ string GetPOS ()
  返回词的词性标注。失败时返回空(NULL)。

- ✓ void SetPOS (const std::string& pos)
  设置词性标注内容；

- ✓ bool HasNE ()
  是否进行了命名实体识别，是返回 true，否则返回 false；

- ✓ string GetNE ()
  返回词的命名实体识别结果。失败时返回空(NULL)。

- ✓ void SetNE (const std::string& ne)
  设置命名实体。

- ✓ bool HasWSD ()
  是否进行了词义消歧，是返回 true，否则返回 false；

- ✓ String GetWSD ()
  返回词义消歧结果。失败时返回空(NULL)。

- ✓ string GetWSDExplanation ()
  返回词义消歧的解释。失败时返回空(NULL)。

- ✓ void SetWSD (const std::string& wsd, const std::string& explanation)
  设置词义消歧；
  参数 wsd 为消歧结果，explanation 为消歧解释；

- ✓ bool HasParser ()
  是否进行了依存句法分析，是返回 true，否则返回 false；

- ✓ int GetParserParent ()
  依存句法分析的父亲结点 ID 号，结返回结果为一个大于等于-2(>= -2)的整数，失败时返回-3。

- ✓ string GetParserRelation ()
  依存句法分析的依存关系。失败时返回空(NULL)。

- ✓ void SetParser (int parent, const std::string& relation)
  设置依存句法；
  参数 parent 为父结点 ID，参数 relation 为依存关系；

- ✓ bool IsPredicate ()
  检查该词是否是谓词。是返回 true，否则返回 false。

- ✓ bool GetSRLs (std::vector<SRL> &srls)
  如果该词是谓词；返回 SRL 类型的 vector<SRL>。SRL 为定义于 Word.h 内的简易封装结构体，其中包括 type(String)，beg(int), end(int)三个公有成员变量。

SRL 类型

该类型是在 Word.h 中声明的；
SRL 类型是对语义角色标注结果的一种抽象，主要包括语义角色标注类型（结果），开始词的 ID 号及结束词的 ID 号

- ✓ type
  公有 String 类型，语义角色标注类型（结果）；

- ✓ beg
  公有 int 类型，开始词的 ID 号；

- ✓ end
  公有 int 类型，结束词的 ID 号；

C++调用示例

下面给出两个例子

例一：

对文本分析；将待分析的文本以字符串的方式给出，并将结果按分词、ID、词性、命名实体、依存关系、词义消歧、语义角色标注的顺序输出。

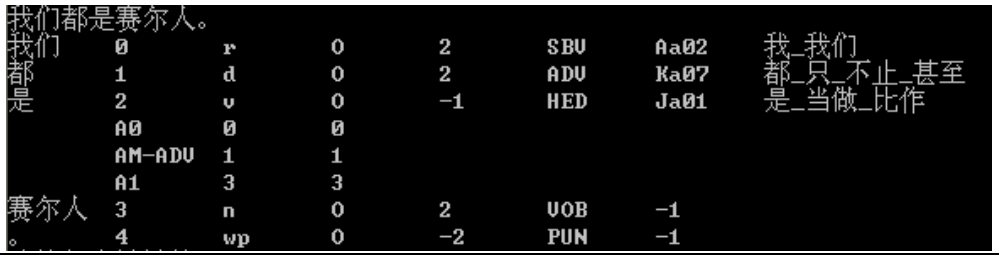
```
using namespace HIT_IR_LTP;
int main(){
	LTPService ls("username:password");
	LTML ltml;
	if (!ls.Analyze(LTPOption.ALL,"我们都是赛尔人。", ltml)) {
		cerr<<"Authorization is denied!"<<endl;
		exit(EXIT_FAILURE);
	}
	int sentNum = ltml.CountSentence();
	for ( int i = 0; i<sentNum; ++i) {
		string sentCont;
		ltml.GetSentenceContent(sentCont, i);
		cout<< sentCont <<endl;
		vector<Word> wordList;
		ltml.GetWords(wordList, i);
		//按句子打印输出
		for( vector<Word>::iterator iter = wordList.begin(); iter!= wordList.end(); ++iter ){
			cout<<iter->GetWS()<<"\t"<<iter->GetID();
			cout<<"\t"<<iter->GetPOS();
			cout<<"\t"<<iter->GetNE();
			cout<<"\t"<<iter->GetParserParent()<<"\t"<<iter->GetParserRelation();
			cout<<"\t"<<iter->GetWSD()<<"\t"<<iter->GetWSDExplanation();
			cout<<endl;
			if( iter->IsPredicate() ){
				vector<SRL> srls;
				iter->GetSRLs(srls);
				for(vector<SRL>::iterator iter = srls.begin(); iter != srls.end(); ++iter){
					cout<<"\t"<<iter->type
						<<"\t"<<iter->beg
						<<"\t"<<iter->end
						<<endl;
				}
			}
		}
	}
	return 0;
```

}

输入为单一文本：

我们都是赛尔人。

输出为：



例二：

首先进行分词，将所得的词按用户词表进行合并或拆分，并对其进行依存句法分析。本例中将“午夜”与“巴赛罗那”进行了合并。

```
using namespace HIT_IR_LTP;
int main(){
	LTPService ls("username:password");
	LTML ltmlBeg;
	try{
		if(!ls.Analyze(LTPOption.WS, "午夜巴塞罗那是对爱情的一次诙谐、充满智慧、独具匠心的冥想。", ltmlBeg))
{
			cerr<<"Authorization is denied!"<<endl;
			exit(EXIT_FAILURE);
		}
		vector<Word> wordList;
		ltmlBeg.GetWords(wordList, 0);
		//输出分词结果
		for( vector<Word>::iterator iter = wordList.begin(); iter!= wordList.end(); ++iter )
		{
			cout<<iter->GetID()<<"\t"<<iter->GetWS()<<endl;
		}
		cout<<endl;

		//将“午夜”与“巴赛罗那”合并，其它的词不变
		vector<Word> mergeList;
		Word mergeWord;
		mergeWord.SetWS(wordList.at(0).GetWS() + wordList.at(1).GetWS());
		mergeList.push_back(mergeWord);

		for (vector<Word>::iterator iter = wordList.begin()+2; iter != wordList.end(); ++iter)
		{
			Word others;
			others.SetWS(iter->GetWS());
			mergeList.push_back(others);
		}
		LTML ltmlSec;
		ltmlSec.AddSentence(mergeList, 0);
		ltmlSec.SetOver();
		LTML ltmlOut;
		ls.Analyze(LTPOption.PARSER, ltmlSec, ltmlOut);

		//输出合并分词后PARSER结果
		cout<<"merge and get parser results."<<endl;
		vector<Word> outList;
		ltmlOut.GetWords(outList,0);
```

```
            for (vector<Word>::iterator iter = outList.begin(); iter != outList.end(); ++iter)
            {
                  cout<<iter->GetID()<<"\t"<<iter->GetWS()<<"\t"<<iter->GetPOS()<<"\t"<<iter->GetParserParent()
<<"\t"<<iter->GetParserRelation()<<endl;
            }
            cout<<endl;
      }catch(exception& e){
            std::cerr<<e.what();
      }
      return 0;
}
```

输入数据为一个完整句子：

午夜巴塞罗那是对爱情的一次诙谐、充满智慧、独具匠心的冥想。

输出为：

```
0       午夜
1       巴塞罗那
2       是
3       对
4       爱情
5       的
6       一
7       次
8       诙谐
9       、
10      充满
11      智慧
12      、
13      独具匠心
14      的
15      冥想
16      。

merge and get parser results.
0       午夜巴塞罗那    nh      1       SBV
1       是      v       -1      HED
2       对      p       9       ADV
3       爱情    n       4       DE
4       的      u       7       ATT
5       一      m       6       QUN
6       次      q       7       ATT
7       诙谐    a       2       POB
8       、      wp      -2      PUN
9       充满    v       1       VV
10      智慧    n       9       VOB
11      、      wp      -2      PUN
12      独具匠心        i       13      DE
13      的      u       14      ATT
14      冥想    v       9       VOB
15      。      wp      -2      PUN
```

JAVA 接口

LTP 需要与服务器进行交互，以完成对文本的分析，分析的返回结果存储在以 DOM 形式组织的 XML 中，客户端接到分析结果后可通过提供的接口对结果进行分析。
需要的工具包：

- ✓ Httpclient.jar
- ✓ Httpcore.jar
- ✓ Jdom.jar（1.1 或更高）
- ✓ Commons-logging.jar

Java 接口的操作主要有以下两个类：
LTPService类为与服务器交互类，负责验证、连接分析参数设置等；
Ltml类为返回数据(XML)解析函数，专门负责数据的提取；

具体接口如下：

LTPService 类

位于 **edu.hit.ir.ltpService .LTPService**
该类主要负责与服务器交互，并将返回结果以 Ltml 对象返回。

- ✓ LTPService(String authorization)
  构造函数，authorization 为用户验证信息，信息格式为：**username:password**；

- ✓ boolean setEncoding(String encodeType)
  设置字符编码，默认为 gbk（LTPOption.GBK），目前仅支持 UTF-8(LTPOption.UTF8)及 GBK，GB2312。编码的定义请参考 edu.hit.ir. LTPOption.java。

- ✓ LTML analyze(String option, String analyzeString)
  参数 option 为分析方式，分析的方式包括：分词（LTPOption.WS），词性标注（LTPOption.POS），命名实体识别（LTPOption. NE），词义消歧（LTPOption.WSD），依存句法分析（LTPOption. PARSER），语义角色标注（LTPOption.SRL）。
  该方法抛出 JDOMException, IOException, RuntimeException 异常，其中 RuntimeException 为用户验证信息未通过认证，请确定您的用户名及密码
  返回对象为 Ltml 类型（各种 set 方法是对分析属性的设置，应在本操作之前）；

- ✓ LTML analyze(String option, LTML ltmlIn)
  参数 option 为分析方式，分析的方式包括：分词（LTPOption.WS），词性标注（LTPOption.POS），命名实体识别（LTPOption. NE），词义消歧（LTPOption.WSD），依存句法（LTPOption. PARSER），语义角色标注（LTPOption.SRL）,全部分析（LTPOption.ALL）。
  参数 ltmlIn 为待分析的 Ltml 对象，其内容可以由用户指定。

该方法抛出 JDOMException, IOException, RuntimeException 异常，其中 RuntimeException 为用户验证信息未通过认证，请确定您的用户名及密码
返回对象为 Ltml 类型（各种 set 方法是对分析属性的设置，应在本操作之前）；

LTML 类

位于 **edu.hit.ir.ltpService.LTML**
该类是对返回的数据(xml)进行解析的主要对象。

- ✓ boolean hasSent ()
  返回结果是否进行了分句。是返回 true，否返回 false。

- ✓ boolean hasWS ()
  返回结果是否进行了分词。是返回 true，否返回 false。

- ✓ boolean hasPOS ()
  返回结果是否进行了词性标注。是返回 true，否返回 false。

- ✓ boolean hasNE ()
  返回结果是否进行了命名实体分析。是返回 true，否返回 false。

- ✓ boolean hasParser ()
  返回结果是否进行了依存句法分析。是返回 true，否返回 false。

- ✓ boolean hasWSD ()
  返回结果是否进行了词义消歧分析。是返回 true，否返回 false。

- ✓ boolean hasSRL ()
  返回结果是否进行了语义角色标注。是返回 true，否返回 false；

- ✓ int countParagraph ()
  分析结果中段落数。

- ✓ int countSentence ()
  分析结果中句子数。

- ✓ int countSentence (int paragraphIdx)
  分析结果中，段落 paragraphIdx 中的句子数；
  参数 paragraphId 为段落号。

- ✓ ArrayList<Word> getWords(int paragraphId, int sentenceId)
  返回分析结果中，第 paragraphId 段的第 sentenceId 句中的所有词 ；
  参数 paragraphId 为段落号，参数 sentenceId 相应段落中的句子号；
  返回类型为 Word 类型的 ArrayList。

- ✓ ArrayList <Word> getWords (int globalSentenceId)
  返回分析结果中，在整篇文章的第 globalSentenceId 名中的所有词；
  参数 globalSentenceId 为整篇文章中句子的序号；
  返回类型为 Word 类型的 ArrayList。

- ✓ String getSentenceContent(int paragraphIdx, int sentenceIdx)
  返回分析结果中，第 paragraphId 段的第 sentenceId 句的内容；
  参数 paragraphId 为段落号，参数 sentenceId 相应段落中的句子号；
  返回类型为 String。

- ✓ String getSentenceContent(int globalSentIdx)
  返回分析结果中，在整篇文章的第 globalSentenceId 句的内容；
  参数 globalSentenceId 为整篇文章中句子的序号；
  返回类型为 String。

- ✓ void saveDom(String filename)
  将 LTML 对象保存到 XML 文件中。
  参数 filename 为文件名。

- ✓ String getXMLStr()
  以字符串的形式返回 LTML。
  返回类型为 String。

- ✓ String getEncoding()
  获得 xml 的字符编码方式
  返回类型为 String

- ✓ void clear()
  将 LTML 对象清空；

以下几个方法是向 Ltml 对象写数据的方法。注意以下几个问题：

1. 请保证调用以下方法的 LTML 对象为空的，或首先调用过 clear 方法，以保证 LTML 对象中数据一致；
2. 输入的数据必须保证一致，例如，如果第一个词完成了词性标注，则其余词也须完成词性标注，因为 LTML 对象是根据第一个词或第一句话生成的 note 结点；
3. 已经调用 setOver 的 LTML 对象不允许调用以下方法，否则会抛异常；凡是由 LTPService 对象返回的 Ltml 对象都调用过 setOver 方法；

- ✓ void setParagraphNumber(int paragraphNumber)
  设置段落数量。注意该方法只能向空的（初始）Ltml 对象中设置，否则会抛出 IllegalArgumentException 异常。
  默认的段落数量为 1

- ✓ void addSentence(ArrayList<Word> wordList, int paragraphIdx)
  向第 paragraphIdx 个段落中增加一个句子；
  参数 wordList 为句子中的词列表；note 结点是根据第一个 sentence 第一个词生成的，必须保证后面句子中的词与第一个词一致，否则会抛异常；
  参数 paragraphIdx 为插入的段落号（从 0 开始）；
  该方法会抛出 DOMException, IOException，IllegalArgumentException，IndexOutOfBoundsException 异常。

- ✓ void addSentence(String sentenceContent, int paragraphIdx)
  向第 paragraphId 个段落中增加一个句子，该方法适用于只完成了分句的情况；
  参数 sentenceContent 为句子内容；参数 paragraphIdx 为插入的段落号（从 0 开始）；
  该方法抛出 DOMException, IOException，IllegalArgumentException，IndexOutOfBoundsException 异常。

- ✓ void setOver()
  一旦调用该方法，LTML 对象就将不能再调用 addSentence 及 setParagraphNumber，每次由 analyze 方法返回的 LTML 对象都会调用该方法；如，在调用 AddSentence 添加完所有的句子后，调用 setOver 表明调用结束；

Word 类型

edu.hit.ir.ltpService.Word
该类型是对 Element 进行的封装，任何的分析结果必须先取到 Word，才能取到相当相应的分析数据。

- ✓ Word()
  构造函数，构造一个空的 Word 对象。

- ✓ boolean hasID ()
  是否有 ID 号，有返回 true，否则返回 false；此 ID 为由服务器端生成的该词在句子中的唯一 ID 号，因此，此 ID 只在句子中才有意义；

- ✓ int getID ()
  返回词的 ID 号；此 ID 为由服务器端生成的该词在句子中的唯一 ID 号，因此，此 ID 只在句子中才有意义；

- ✓ void setID (int id)
  设置词在句子中的 ID 号；

- ✓ boolean hasWS ()
  是否进行了分词，是返回 true，否则返回 false；

- ✓ String getWS ()

返回分词的具体内容；

- ✓ void setWS (String content)
  设置分词内容；

- ✓ boolean hasPOS ()
  是否进行了词性标注，是返回 true，否则返回 false；

- ✓ String getPOS ()
  返回词的词性标注；

- ✓ void setPOS (String pos)
  设置词性标注内容；

- ✓ boolean hasNE ()
  是否进行了命名实体识别，是返回 true，否则返回 false；

- ✓ String getNE ()
  返回词的命名实体识别结果。

- ✓ void setNE (String ne)
  设置命名实体。

- ✓ boolean hasWSD ()
  是否进行了词义消歧，是返回 true，否则返回 false；

- ✓ String getWSD ()
  返回词义消歧结果。

- ✓ String getWSDExplanation ()
  返回词义消歧解释；

- ✓ void setWSD (String wsd, String explanation)
  设置词义消歧；
  参数 wsd 为消歧结果，explation 为消歧解释；

- ✓ boolean hasParser ()
  是否进行了依存句法分析，是返回 true，否则返回 false；

- ✓ int getParserParent ()
  依存句法分析的父亲结点 ID 号，结返回结果为一个大于等于-2(>= -2)的整数，失败时返回-3；

- ✓ String getParserRelation ()

依存句法分析的依存关系；

- ✓ void setParser (int parent, String relation)
  设置依存句法；
  参数 parent 为父结点 ID，参数 relation 为依存关系；

- ✓ boolean isPredicate ()
  检查该词是否是谓词。是返回 true，否则返回 false；

- ✓ ArrayList<SRL> getSRLs()
  如果该词存在 SRL 分析结果。返回 SRL 类型的 List。SRL 为定义于 edu.hit.ir.ltpService.SRL 的简易封装类型，其中包括 type(String)，beg(int), end(int)三个成员变量（公有 final 类型）。

SRL 类型

edu.hit.ir.ltpService.SRL
SRL 类型是对语义角色标注结果的一种抽象，主要包括语义角色标注类型（结果），开始词的 ID 号及结束词的 ID 号

- ✓ type
  公有 final String 类型，语义角色标注类型（结果）；

- ✓ beg
  公有 final int 类型，开始词的 ID 号；

- ✓ end
  公有 final int 类型，结束词的 ID 号；

JAVA 调用示例

下面给出两个例子：

例一：

对文本分析；将待分析的文本以字符串的方式给出，并将结果按分词、ID、词性、命名实体、依存关系、词义消歧、语义角色标注的顺序输出。

```
    public static void main(String[] args) {
        LTPService ls = new LTPService("username:password");
        try {
            LTML ltml = ls.analyze(LTPOption.ALL,"我们都是赛尔人。");
            int sentNum = ltml.countSentence();
            for(int i = 0; i< sentNum; ++i){
//                逐句访问
                ArrayList<Word> wordList = ltml.getWords(i);
                System.out.println(ltml.getSentenceContent(i));
                for(int j = 0; j < wordList.size(); ++j){
                    System.out.print("\t" + wordList.get(j).getWS());
                    System.out.print("\t" + wordList.get(j).getPOS());
                    System.out.print("\t" + wordList.get(j).getNE());
                    System.out.print("\t" + wordList.get(j).getWSD() + "\t" +
wordList.get(j).getWSDExplanation());
                    System.out.print("\t" + wordList.get(j).getParserParent() + "\t" +
wordList.get(j).getParserRelation());
//                    如果是谓词则输出
                    if(ltml.hasSRL() && wordList.get(j).isPredicate()){
                        ArrayList<SRL> srls = wordList.get(j).getSRLs();
                        System.out.println();
                        for(int k = 0; k <srls.size(); ++k){
                            System.out.println("\t\t" + srls.get(k).type + "\t" +
srls.get(k).beg + "\t" + srls.get(k).end);
                        }
                    }
                    System.out.println();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

输入为单一文本：

我们都是赛尔人。

输出为：

我们都是赛尔人。

```
    我们r   O   Aa02   我_我们 2   SBV
    都  d   O   Ka07   都_只_不止_甚至 2   ADV
    是  v   O   Ja01   是_当做_比作-1  HED
        A0  0   0
        AM-ADV 1   1
        A1  3   3

    赛尔人  n   O   -1      2   VOB
    。  wp  O   -1      -2  PUN
```

例二：

首先进行分词，将所得的词按用户词表进行合并或拆分，并对其进行依存句法分析。本例中将“午夜”与“巴赛罗那”进行了合并。

```
    public static void main(String[] args) {
        LTPService ls = new LTPService("username:password");
        try {
            LTML ltmlBeg = ls.analyze(LTPOption.WS,"午夜巴塞罗那是对爱情的一次诙谐、充满智慧、独具匠心的冥想。");
            LTML ltmlSec = new LTML();
            int sentNum = ltmlBeg.countSentence();
            for(int i = 0; i< sentNum; ++i){
                ArrayList<Word> wordList = ltmlBeg.getWords(i);
                for(int j = 0; j < wordList.size(); ++j){
                    System.out.print("\t" + wordList.get(j).getID());
                    System.out.print("\t" + wordList.get(j).getWS());
                    System.out.println();
                }
//                  merge
                ArrayList<Word> mergeList = new ArrayList<Word>();
                Word mergeWord = new Word();
                mergeWord.setWS(wordList.get(0).getWS()+wordList.get(1).getWS());
                mergeList.add(mergeWord);
                for(int j = 2; j < wordList.size(); ++j){
                    Word others = new Word();
                    others.setWS(wordList.get(j).getWS());
                    mergeList.add(others);
                }
                ltmlSec.addSentence(mergeList, 0);
            }
            ltmlSec.setOver();
            System.out.println("\nmerge and get parser results.");
            LTML ltmlOut = ls.analyze(LTPOption.PARSER, ltmlSec);
            for(int i = 0; i< sentNum; ++i){
                ArrayList<Word> wordList = ltmlOut.getWords(i);
                for(int j = 0; j < wordList.size(); ++j){
                    System.out.print("\t" + wordList.get(j).getID());
                    System.out.print("\t" + wordList.get(j).getWS());
                    System.out.print("\t" + wordList.get(j).getPOS());
                    System.out.print("\t" + wordList.get(j).getParserParent() + "\t" + wordList.get(j).getParserRelation());
                    System.out.println();
                }
            }
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

输入数据为一句话：
午夜巴塞罗那是对爱情的一次诙谐、充满智慧、独具匠心的冥想。

最终输出为：

```
    0   午夜
    1   巴塞罗那
    2   是
    3   对
    4   爱情
    5   的
    6   一
    7   次
    8   诙谐
    9   、
    10  充满
    11  智慧
    12  、
    13  独具匠心
    14  的
    15  冥想
    16  。

merge and get parser results.
    0   午夜巴塞罗那 nh  1   SBV
    1   是  v   -1  HED
    2   对  p   9   ADV
    3   爱情 n   4   DE
    4   的  u   7   ATT
    5   一  m   6   QUN
    6   次  q   7   ATT
    7   诙谐 a   2   POB
    8   、  wp  -2  PUN
    9   充满 v   1   VV
    10  智慧 n   9   VOB
    11  、  wp  -2  PUN
    12  独具匠心 i   13  DE
    13  的  u   14  ATT
    14  冥想 v   9   VOB
    15  。  wp  -2  PUN
```

FAQ

1. [JAVA 版][LINUX][ECLIPSE]
   在 analyze 时，抛出 JDOMParseException，位置可能会在 org.jdom.input.SAXBuilder.build 时，异常内容一般为“must not contain the '<' character”或与之类似问题：
   [解决]
   一般是字符编码方式在 IDE 中不能正确识别造成的，建议统一改为 GBK。在 ECLIPSE 环境可如下设置：window->preferences->General->Workspace，更改 Text file ecoding 为 GBK，或项目->属性->Resource，更改 Text file ecoding 为 GBK
2. [编码]LTP SERVICE 接口支持什么编码方式？
   [解决]
   目前 LTP SERVICE 支持 GBK、GB2312、UTF-8（输入），但建议采用 GBK 编码方式。如果采用 UTF-8 方式，在服务器分析前，会有转码工作（转化为 GBK 编码）；事实上，无论采用什么编码方式，服务器返回的都是 GBK 编码方式，如果此时仍然需要 UTF-8 或其它格式，用户可自行转化。
3. [多线程]目前 LTP 是否支持多线程？
   [解决]
   目前 LTP SERVICE 不支持多线程访问，为了服务的持续稳定，请不要使用多线程访问。