

Windows の Python で JUMAN++ を 使う

きた@satamame

株式会社リーディング・エッジ社

概要

Windows の Python で JUMAN++ を使う時の注意点をまとめました。

1. ソースからビルドする必要がある
2. コマンドとして実行可能にする
3. Python からの呼び出し方
4. 形態素解析できないケースがある
5. エラーを減らすための工夫をする

※ JUMAN++ は、高性能な日本語形態素解析システムです。

環境

- Windows 10 Home
- Python 3.8.1
- JUMAN++ 2.0.0-rc3
- pyknp 0.4.1
- Visual Studio Community 2017

ソースからビルドする

ソースからビルドする (1)

- Windows 用のバイナリは配布されていない。
- でも、無料の Visual Studio でビルドできる。
- しかも、丁寧に解説してくれる記事がある。

ソースからビルドする(2)

"JUMAN++v2 Windows ビルド" で検索すると良い記事があるので、その通りやります。

The screenshot shows a search results page with the query "JUMAN++v2 Windows ビルド" in the search bar. The results are filtered to show "すべて" (All) items. There are approximately 191,000 results found in 0.42 seconds.

Top result:

- [tadaoyamaoka.hatenablog.com › entry › 2019/07/26](https://tadaoyamaoka.hatenablog.com/entry/2019/07/26)
- Juman++v2をWindowsでビルドする - TadaoYamaokaの日記**
- 2019/07/26 - Juman++v1はWindowsに対応していなかったが、v2は公式でWindowsにも対応している。現在rc2がリリースされているが、ビルド済みバイナリは配布されていないため自分でビルドする必要がある。 rc2 ...

Second result:

- [qiita.com › mecab](https://qiita.com/mecab/items)
- NLP準備運動：分かち書き環境の構築 Mecab , Juman++ver2 ...**
- 2019/03/17 - ※Juman++ Version.2の使用のためには「Visual Studio2017」と「cmake」が必要になります。 ... Terminalからのmecab-python-0.996フォルダで「python setup.py build」を実行します; 正常にbuildが完了したら、「python setup.py install」 ...

ソースからビルドする(3)

ビルドしたディレクトリで、正しい引数をつければ実行できます。

```
> chcp 65001
Active code page: 65001

> echo 今日もいい天気 | .\jumanpp_v2 --config=[path]\jumandic.conf
今日 きょう 今日 名詞 6 時相名詞 10 * 0 * 0 "代表表記:今日/きょう カテゴリ:
時間"
も も も 助詞 9 副助詞 2 * 0 * 0 NIL
いい いい いい 形容詞 3 * 0 イ形容詞イ段 19 基本形 2 "代表表記:良い/よい 反
義:形容詞:悪い/わるい"
天気 てんき 天気 名詞 6 普通名詞 1 * 0 * 0 "代表表記:天気/てんき カテゴリ:
抽象物"
EOS
```

コマンドとして実行可能にする

コマンドとして実行可能にする(1)

以下の3つのファイルを管理しやすいフォルダにコピーします。
(ここまで説明してくれている記事もありました。)

- jumanpp_v2.exe
- jumandic.conf
- jumandic.jppmdl

コマンドとして実行可能にする(2)

たとえば、以下のように配置します。
(このスライドではこの例に沿って説明します。)

```
C:\ProgramData/  
└ jumanpp/  
    └ jumanpp_v2.exe  
└ model/  
    └ jumandic.conf  
    └ jumandic.jppmdl
```

※ Program Files など、スペースを含むパスだと動きません。

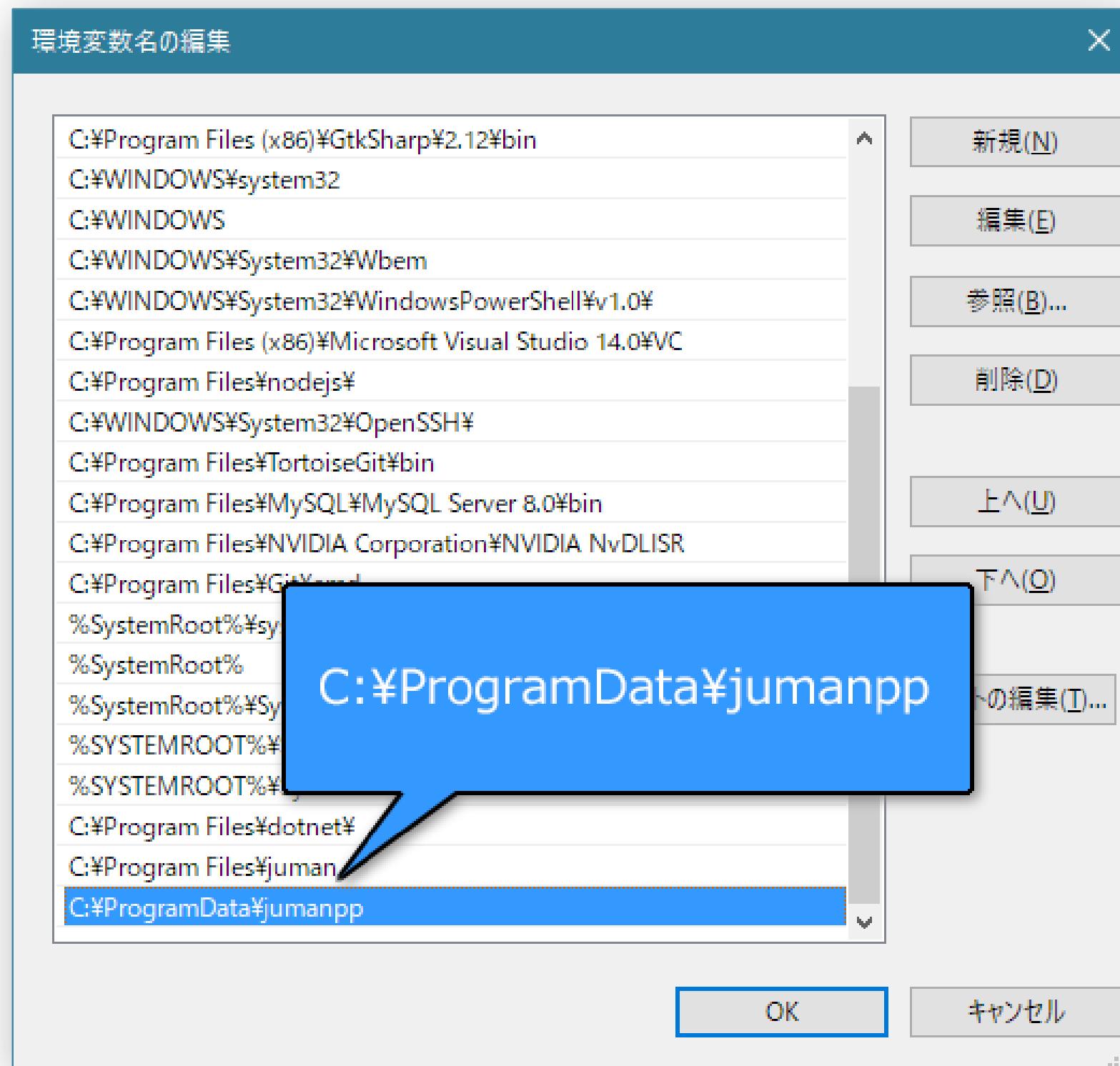
コマンドとして実行可能にする(3)

jumandic.conf の1行目のパスを jumandic.jppmdl へのフルパスにします。

```
--model=C:\ProgramData\jumanpp\model\jumandic.jppmdl  
--rnn-nce-bias=5.62844432562  
--rnn-unk-constant=-3.4748115191  
--rnn-unk-length=-2.92994951022  
--feature-weight-perceptron=1  
--feature-weight-rnn=0.0176
```

コマンドとして実行可能にする(4)

jumanpp_v2.exeのあるディレクトリにパスを通します。



コマンドとして実行可能にする(5)

jumanpp_v2.exeのあるディレクトリに jumanpp.bat を作ります。

```
C:\ProgramData/  
└ jumanpp/  
    ├ jumanpp.bat      <- これを作る  
    ├ jumanpp_v2.exe  
    └ model/  
        ├ jumandic.conf  
        └ jumandic.jppmdl
```

jumanpp.bat の内容

```
@echo off  
jumanpp_v2 --config=C:\ProgramData\jumanpp\model\jumandic.conf %*
```

これで毎回 config を指定せずに実行できるようになります。

コマンドとして実行可能にする(6)

どこかのディレクトリからでも、以下のように実行できます。

```
> chcp 65001
Active code page: 65001

> echo 明日があるさ | jumanpp
明日 あす 明日 名詞 6 時相名詞 10 * 0 * 0 "代表表記:明日/あす カテゴリ:時間"
が が が 助詞 9 格助詞 1 * 0 * 0 NIL
ある ある ある 動詞 2 * 0 子音動詞ラ行 10 基本形 2 "代表表記:有る/ある 反義:
形容詞:無い/ない 補文ト"
さ さ さ 助詞 9 終助詞 4 * 0 * 0 NIL
EOS
```

Python から呼び出す

Python から呼び出す(1)

pyknp をインストールします。

```
> pip install pyknp
```

このパッケージに JUMAN++ を呼び出すクラスが入っています。

Python から呼び出す(2)

1. .bat ファイルはコマンドとして認識されない。
2. インスタンス生成時、コマンドと引数を指定できる。

```
from pyknp import Juman

juman = Juman(command='jumanpp_v2',
               option='--config=C:\ProgramData\jumanpp\model\jumandic.conf')

mrphs = juman.analysis('明日もお休みさ')
for m in mrphs:
    print(m.midasi, m.genkei, m.hinsi, m.bunrui, m.katuyou2)
```

明日 明日 名詞 時相名詞 *

も も 助詞 副助詞 *

お お 接頭辞 名詞接頭辞 *

休み 休む 動詞 * 基本連用形

さ さ 接尾辞 名詞性述語接尾辞 *

形態素解析できないケース

形態素解析できないケース(1)

入力文字列が4096バイトを超えると例外が発生します。

```
s = 'も' * 1366
print(f'Length: {len(s.encode())} bytes')

try:
    mrphs = juman.analysis(s)
except Exception as e:
    print(f'Exception: {e}')
```

```
Length: 4098 bytes
Exception: invalid literal for int() with base 10: 'input'
```

形態素解析できないケース (2)

半角スペースを含む単語 (顔文字など) で例外が発生します。

```
try:  
    mrphs = juman.analysis('m(_ _)m')  
except Exception as e:  
    print(f'Exception: {e}')
```

```
Exception: invalid literal for int() with base 10: 'm(_ '
```

形態素解析できないケース (3)

'\x1a' を形態素解析すると、フリーズします。

```
juman.analysis(' \x1a ')
```

- ※強制終了するしかなくなるので、実行しないで下さい。
- ※でも普通の文書にこんな行はないと思います。

形態素解析できないケース (4)

半角スペース + 小さい「つ」で例外が発生します。

```
try:  
    mrphs = juman.analysis(' つ')  
except Exception as e:  
    print(f'Exception: {e}')
```

```
Exception: invalid literal for int() with base 10: '特殊'
```

形態素解析できないケース (5)

全角スペース + 小さい「つ」が「空白」になります。

```
mrphs = juman.analysis(' つ')
for m in mrphs:
    print(m.midasi, m.genkei, m.hinsi, m.bunrui, m.katuyou2)
```

```
つ      特殊 空白 *
```

※「 って」等と続く場合は正しく解析されるので、普通の文書であればこの問題は起きないと思います。

エラーを減らすための工夫

エラーを減らすための工夫 (1)

pyknp の Juman クラスは、juman_lines() メソッドで OS のコマンドから文字列を受け取っています。

直接実行するところな感じ。

```
juman.juman_lines('m(_ _)m')
```

```
'm(_ _)m m(_ _)m m(_ _)m 特殊 1 記号 5 * 0 * 0 "代表表記:顔文字/顔文字  
顔文字"\n'
```

※ この文字列が半角スペースを区切り文字に使っている。

※ v2.0.0-rc3 で、出力のエスケープの仕方が変わったようです。

エラーを減らすための工夫 (2)

Juman のサブクラスを作って、`juman_lines()` メソッドをオーバーライドすることにしました。

```
import re
from pyknp import Juman

class JumanPsc(Juman):
    # 無効な文字（あとで使う）
    invalid_chars = ['\x1a']

    # 文末文字のパターン（あとで使う）
    end_pattern = r'[。？！]'

    def juman_lines(self, input_str):
        # ここを今から作る
```

エラーを減らすための工夫 (3)

入力文字列から無効な文字を削除します。

```
def juman_lines(self, input_str):  
  
    # 無効な文字を削除する。  
    for c in JumanPsc.invalid_chars:  
        input_str = input_str.replace(c, '')
```

半角スペースを全角にします。

```
# 連続しない半角スペースを全角スペースにする  
input_str = re.sub(r'(?<!\s) (?!\s)', ' ', input_str)
```

※基本的には JUMAN++ が半角文字を全角にして返すが、他の文字と組み合わさった場合に半角が残ることがある。

エラーを減らすための工夫 (4)

スペースの後に「っ」があったら分割して処理するようにします。

分割したそれぞれの文字列でメソッドを再帰的に呼びます。

```
# 空白文字の直後に小さい「つ」があったら、空白文字と「つ」の間で分割
matchObj = re.search(r'\sっ', input_str)
if matchObj:
    # 空白文字までと「つ」以降に分ける。
    s1 = input_str[:matchObj.start() + 1]
    s2 = input_str[matchObj.start() + 1:]

    # それぞれ形態素解析して、結果をつなげて返す。
    return self.juman_lines(s1) + self.juman_lines(s2)
```

エラーを減らすための工夫 (5)

4096バイトを超える場合も、先程と同様に分割して処理するようになります。

```
# 4096バイトを超えるなら、文末文字（「。」等）で分割して形態素解析する。
if len(input_str.encode()) > 4096:
    # 文末文字を探す。
    matchObj = re.search(JumanPsc.end_pattern, input_str)

    # 見つかったら分割して形態素解析してつなげて返す。
    if matchObj:
        s1 = input_str[:matchObj.start() + 1]
        s2 = input_str[matchObj.start() + 1:]
        return self.juman_lines(s1) + self.juman_lines(s2)

    # 分割処理しない（できない）場合は継承元に渡す。
    return super().juman_lines(input_str)
```

<https://github.com/satamame/pscn>