

# PyTCで学ぶ Python C拡張の書き方

末永 匡

a.k.a.

グニャラくん

# 自己紹介

- グニャラくんって申します
  - <http://d.hatena.ne.jp/tasukuchan/>
- 全文検索エンジンの開発とかやっています

# PyTCとは

- Tokyo CabinetのPythonバインディング
- Tokyo Cabinet
  - key-value型データベースの良い実装
    - Berkeley DBとか
    - 要は、永続化できる辞書のようなもの
  - QDBM/Hyper Estraierの作者mikio氏がmixiの支援を受けて開発
  - C言語によって書かれたライブラリ

# こんな風に使えます

```
1 import pytc
2
3 db = pytc.HDB('hdb.db', pytc.HDBOWRITER | pytc.HDBOCREAT)
4 db['niku'] = 'umai'
5 print db['niku']
6
7 db['ra-men'] = 'kuitai'
8 print db['ra-men']
9
10 for key in db:
11     print 'key:', key, ' value:', db[key]
```

# パフォーマンス

- Berkeley DBよりずっと早い
- CDBにはさすがに負ける
  - CDBはデータ更新ができない

	Write			Read		
	Real	User	Sys	Real	User	Sys
Berkeley DB	25.679s	13.754s	6.946s	13.858s	10.492s	3.300s
TC BTree	9.947s	6.340s	2.881s	9.258s	6.499s	2.643s
TC Hash	18.427s	6.377s	7.234s	12.335s	7.177s	5.101s
CDB	8.236s	4.916s	3.154s	7.379s	6.081s	1.269s

<http://plaza.rakuten.co.jp/kugutsushi/diary/200711220000/> より

# 選択肢

- Pythonで外部のライブラリを使用するための主な手段
  - ctypes
  - SWIG
  - Pyrex
  - SIP
  - Boost.Python
  - Cによる拡張モジュール

# ctypes

- Windowsではおなじみ
  - Python 2.5以降は標準モジュール
- お手軽・コンパイルいらず
  - お試しで使う場合、パフォーマンスを気にしない場合にはオススメ！
- 実用的にするにはラッパをPythonで書く必要あり
  - dictみたいに使いたいよね
  - `__getitem__`, `__setitem__`, `__delitem__`とか

# SWIG

- 移植性が高い
  - perl/python/ruby/Java etc.のバインディングが一挙に書ける
- C++とSWIGは相性いいです。
  - CでのAPIだと、オブジェクトの作成・削除については自分で書かないといけない
  - C++ラッパを書いてSWIGにかけるとよい
- けどどもだっけえ〜ど

# SWIG(2)

- 移植性が必要なかった
  - すでにPerl/Ruby/Javaバインディングが存在する
- C++のラッパを書くのが面倒だった
- コールバックが書きにくかった
  - 移植性を捨てれば自由に書けるが...嬉しくない
- Pythonで書いたラッパが実用上必要
  - `__getitem__`, `__setitem__`, `__delitem__`とか
- あんまり嬉しくない。。

# Pyrex

- Python+  $\alpha$  の文法で拡張モジュールが書ける
  - Cのソースが出力される
- 書きやすさ・パフォーマンスに優れる
  - コールバックも問題なし
- オススメ！
  - 普通はコレを使おう！

# SIP/Boost.Python

- SIP
  - QtのPythonバインディングであるPyQtを書くために作られたツール
    - ちゃんと評価してません。。
- Boost.Python
  - C++ライブラリBoostの一部
  - 以下の点で採用を見送りました
    - C++でラッパを書く必要がある
    - ビルドにBoostが必要

# Cによる拡張モジュール

- Python/C APIを使って書く
- Cで全て書ける
  - 大変面倒
  - 自由度とパフォーマンスに勝る
- 素人にはオススメできない

# どれを選んだか

- Cによる拡張モジュールを選ぶ
  - 趣味だから！
  - Tokyo Cabinetのパフォーマンスへのあくなき欲求に敬意を払って

# Cで書くときの方針

- Pythonでのラッパを必要としないように
  - 1つのCのソースと、setup.pyのみ
  - パッケージ構成の単純化・高速化
- マクロを活用
  - 同じような引数をとって、同じような引数を返す関数をマクロ化
  - コーディングの省力化

# dictみたいなオブジェクトを作る

- Pythonでラッパを書けば簡単
- Cで書く場合には、以下のメンバ群を設定し、実装を行う。
  - tp\_as\_mapping
    - \_\_getitem\_\_, \_\_setitem\_\_, \_\_delitem\_\_
  - tp\_iter
    - \_\_iter\_\_
  - tp\_as\_sequence(忘れがち)
    - \_\_contains\_\_

# マルチスレッド利用のために

- Tokyo CabinetはI/O操作を行う
  - ブロックの可能性
- グローバルインタプリタロック(GIL)を開放して、マルチスレッド時での同時実行性能を上げる
  - `Py_BEGIN_ALLOW_THREADS`
  - `Py_END_ALLOW_THREADS`
- コールバック関数ではGILを取得
  - `PyGILState_Ensure()/_Release()`

# 定数の格納

- 現状: モジュールそのものの辞書に登録
  - ex.) `pytc.HDBOCREAT`
- 将来的には、型に定数を登録したい
  - ex.) `pytc.HDB.OCREAT`
  - `__getattr__` ?
  - `__slots__` ?

# まとめ

- お気軽にctypes
- C++のラッパがあるんだったらSWIGも
- ちゃんと書くときはPyrex
- 変態で暇ならC APIでゴリゴリ書くと楽しいよ

# 今後の予定

- iterkeys(), itervalues(), iteritems()も実装
- みんな使ってね！
  
- リポジトリは  
<http://svn.coderepos.org/share/lang/python/pytc/>
- pypiでは  
<http://pypi.python.org/pypi/pytc/>