Tokyo Cabinet

@hiratara
FreakOut, Inc.

FreakOut について

- https://www.fout.co.jp/
- 広告を主軸に色々やってます
- がっつり Perl を使っています
- Tokyo から来ました

Tokyo Cabinet について

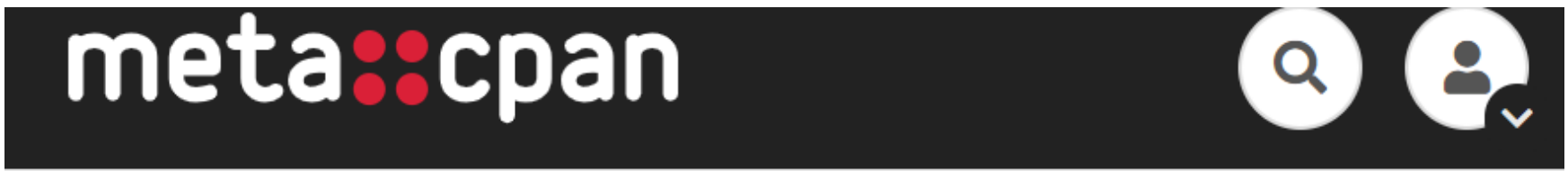
- http://fallabs.com/tokyocabinet/
- **東側のプロダクトです**
- Mikio さんが 15 年前に開発した DBM
  - 開発はすでに終了している
  - Hash, B+ Tree, Fixed-length, Table の実装
- 姉妹プロダクト
  - QDBM
  - Kyoto Cabinet
  - 🆕 Tkrzw

Tokyo Cabinet と私 (1)

- オフラインデータベースとして利用
  - Hash データベースを中心に利用
  - 必要な情報を出力
  - web server の足元に配置し、高速アクセス

Tokyo Cabinet と私 (2)

- めちゃくちゃ速い
  - Perl のハッシュのキーアクセスに毛が生えた程度
- 扱いがすごく楽
  - ただのファイルなのでコピーするだけ
- 同梱の Perl binding がある



No search results for tkrzw

Tokyo Cabinet と私 (3)

- 約 30 種類のデータベースファイル
- 数百 KB ～数十 GB のサイズ
- 数千万件のデータ
- チューニング法など、中身をよく知りたい

Rust で再実装

https://github.com/hiratara/rs-tchread

- 自分が長年使っているものをよく知りたい
- Rust を書く練習をしたい
- 「ハッシュデータベース」の「読み込み」に的を絞って実装
  - `tchmgr` の `get` `list` 辺りの機能が目標
  - せっかくならオマケ機能も

Tokyo Cabinet の仕様

- ドキュメント によく書かれている
  - 実装には色々なノウハウが詰め込まれているが、ファイルフォーマットは非常にシンプル
- mixi engineer blog

マジックナンバ	0	32	データ
データベースタイプ	32	1	ハッシ
追加フラグ	33	1	開きっ
アラインメント力	34	1	アライ
フリーブロックプールカ	35	1	フリー
オプション	36	1	ラージ 外部圧
バケット数	40	8	バケッ
レコード数	48	8	格納し
ファイルサイズ	56	8	データ
先頭レコード	64	8	最初の
不透明領域	128	128	ユーザ

Rust の良い点 (1)

(Cと比べて) ジェネリクスが強力

- `R` → file/on memory に同時に対応
- `B` → 32bit/64bit に同時に対応
- Zero Cost Abstractions: コンパイル時に展開

```
pub struct TCHDB<B, R> {
    pub reader: R,
    pub header: Header,
    pub bucket_offset: u64, // always be 256
    pub free_block_pool_offset: u64,
    bucket_type: PhantomData<fn() -> B>,
}
```

Rust の良い点 (2)

crates.io の資源が使える

- binrw
  - バイナリデータの高速な読み書き
  - わかりやすい宣言的な記述
- structopt
  - コマンドライン引数の宣言的なパース

```
#[derive(BinRead, Debug)]
#[br(little)]
pub struct Header {
    #[br(count = 32, assert(magic_number.starts_with(b"ToK
    pub magic_number: Vec<u8>,
    #[br(assert(database_type == 0))]
    pub database_type: u8,
    pub additional_flags: u8,
    pub alignment_power: u8,
    pub free_block_pool_power: u8,
    #[br(pad_after = 3)]
    pub options: u8,
    pub bucket_number: u64,
    pub record_number: u64,
    pub file_size: u64,
    #[br(pad_after = 56)]
    pub first_record: u64,
    #[br(count = 128)]
    pub opaque_region: Vec<u8>,
}
```

Rust の良い点 (3)

動作が C 並に高速

```
$ time perl tchcount.pl casket.tch
1680800

real    0m2.273s
user    0m1.774s
sys     0m0.499s

$ time tchmgr list -nl casket.tch | wc -l
1680800

real    0m1.259s
user    0m0.609s
sys     0m0.746s

$ time rs-tchread list casket.tch | wc -l
1680800

real    0m0.631s
user    0m0.618s
sys     0m0.053s
```

おまけ機能 (1)

統計情報（バケットの利用状況、パディング長など）。

```
$ rs-tchread inspect casket.tch
# of buckets: 100663291
# of empty buckets: 42791771
# of records: 86118651
# of records without children: 60198570
# of records with one child: 23593031
# of records with two children: 2327050
avg of key length: 27.00043954473927
avg of value length: 141.19580666678115
avg of padding length: 8.889789228119701
# of free blocks: 0
```

おまけ機能 (2)

バケットの dump 。

```
$ rs-tchread dump-bucket casket.tch 1
record 1: hash=129, key=p
record 2: hash=131, key=r
record 3: hash=133, key=t
record 4: hash=135, key=v
record 5: hash=139, key=z
record 6: hash=147, key=b
record 7: hash=149, key=d
record 8: hash=151, key=f
record 9: hash=153, key=h
record 10: hash=155, key=j
record 11: hash=157, key=l
record 12: hash=159, key=n
```

キーの深さ。

```
$ rs-tchread trace-to-get casket.tch h
bucket: 1
hash: 153
record 1: hash=147, key=b
record 2: hash=149, key=d
record 3: hash=151, key=f
record 4: hash=153, key=h
7
```

おまけ機能 (3)

`rs-tchread --bigendian`

- aptで入るdebian/ubuntuの Tokyo Cabinet は壊れている
  - endian が逆で、他の環境で生成したDBとの互換がない
- `--enable-swab` オプションは big endian 環境では動かない
- Bug#667979: libtokyocabinet9: TokyoCabinet got endianness in DB wrong on both big- and little-endian architectures
  - 11 years ago

まとめ

- 開発が終わった OSS が活躍していることもある（自己責任）
- Rust はいいぞ
- 再実装は勉強になる