多様性と戦う

Fight with Diversity

田中 哲
産業技術総合研究所 (AIST) /
フリーソフトウェアイニシアティブ (FSIJ)
RubyKaigi 2013
2013-06-01

RubyKaigi 2008 テーマ

多様性

google検索

「RubyKaigi 2008 多様性　感動」

2890件

ところで、

ブラウザの多様性に困ったことはありませんか(例:IE6)

「多様性は善」というのはちょっといいすぎかも

今日の話

多様性は敵だ

Ruby を開発する

- デザインする
  - 用法を調べる
  - 良いやり方を発見する
- 実装する
  - 手元で作って動かす
  - 様々な環境に対応させる　　←　今日の話

標準添付拡張ライブラリ dbm

- Unix に古くからあるデータベースライブラリへのインターフェース
- 永続的なハッシュテーブル
- キーと値は文字列のみ

dbm の使い方

- ファイルにデータを記録する

```
require 'dbm'
DBM.open("dbfile", DBM::WRITER) {|d|
  d["foo"] = "bar"
}
```

- 他のプロセスはデータを読み出せる

```
require 'dbm'
DBM.open("dbfile", DBM::WRITER) {|d|
  p d["foo"]          #=> "bar"
}
```

dbm がサポートするライブラリ

4種類のライブラリをサポート
Ruby ビルド時に見つかったものを使う

- ndbm: New Database Manager
- gdbm: GNU dbm
- db: Berkeley DB
- qdbm: Quick Database Manager

Ruby の拡張ライブラリ

Ruby のディレクトリ構成

- ruby　　Ruby 本体
  - ext　　拡張ライブラリ
    - dbm　　dbm拡張ライブラリ
      - extconf.rb　Makefile生成スクリプト
      - dbm.c　　実装

extconf.rb

- 環境を調べ、Makefile を生成するスクリプト
- configure みたいなもの

単純な extconf.rb

fcntl/extconf.rb は 2行

```
require 'mkmf'
create_makefile('fcntl')
```

- mkmf というライブラリを require する
- fcntl 拡張ライブラリのために Makefile を生成

mkmf

mkmf は extconf.rb で使うライブラリ

- create_makefile　Makefile の生成
- have_library　C のライブラリの存在確認
- have_type　C の型の存在確認
- have_func　C の関数の存在確認
- など

dbm の extconf.rb

けっこう長い

- ext/curses/extconf.rb　116行
- ext/openssl/extconf.rb　157行
- ext/dbm/extconf.rb　254行 ←
- ext/socket/extconf.rb　525行
- ext/tk/extconf.rb　2057行

(Ruby 2.0.0)

C 言語の dbm の歴史

- 1979: Unix Version 7 が dbm を提供
- 1986: 4.3BSD が ndbm を提供
- 1990: gdbm 1.1 リリース
- 1991: Net/2 に Berkeley DB が同梱
- 2003: qdbm 1.1.0 リリース

Unix Version 7 の dbm (1979)

- AT&T 提供で AT&T ライセンスが必要
- プロセスでひとつのデータベースしか扱えない
- ファイルは .pag と .dir のふたつ
- キーと値のサイズに制限あり
  両方あわせて 512バイト
  同じハッシュ値のキーがあるなら、それらすべてあわせて 512バイト
  (4.2BSD では 1024 バイトに拡大)
- Ruby の dbm はサポートしていない

Unix Version 7 の dbm (1979)

- API: dbm(3x) からの抜粋

  typedef struct { char *dptr; int dsize; } datum;
  int dbminit(file)
  datum fetch(key)
  void store(key, content)

- ヘッダは提供されていない?
  (ソース中にはある)
- ライブラリは libdbm (-ldbm として使う)
- 構造体を引数と返値に使っている

4.3BSD の ndbm (1986)

- Unix Version 7 の dbm からの派生
  AT&T ライセンス
- 複数のデータベースを扱える
- ファイルは .pag と .dir のふたつ
- 後に標準化された (Single Unix Specification)
- Ruby の dbm が想定する API

4.3BSD の ndbm (1986)

- API: ndbm(3) からの抜粋
  #include <ndbm.h>
  DBM *dbm_open(file, flags, mode)
  void dbm_close(db)
  datum dbm_fetch(db, key)
  int dbm_store(db, key, content, flags)
- 関数名は dbm_ で始まる
- libc に入っているのでリンカへの指定は不要

gdbm (1990)

- GNU オペレーティングシステム (自由で Unix ライクな OS) には AT&T ライセンスでない 自由な dbm/ndbm 互換ライブラリが必要
- gdbm 独自の API に加え dbm, ndbm 互換 API を持つ
- 互換API でのファイルは .pag と .dir のふたつ ただしハードリンクされて実体はひとつ
- gdbm 1.9 からはハードリンクされず、.dir にはバージョン情報が格納される

gdbm (1990)

- ヘッダは gdbm.h と ndbm.h を提供
- ライブラリは libgdbm
- gdbm 1.8.1 以降は libgdbm_compat が分離
  互換関数をリンクしないことが可能になった
- gdbm 独自の関数名は gdbm_ で始まる

Berkeley DB (1991)

- Berkeley でも AT&T ライセンスが不要な OS をリリースする活動があり、Net/2 で ndbm の代替として Berkeley DB が提供された
- dbm の代替は後で追加 (4.4BSD Alpha, 1992)
- 独自の API もあり高機能 (B木など)
- ファイルは .db のひとつ
- ヘッダは db.h と ndbm.h を提供
- libc に入っているのでリンカへの指定は不要
- Berkeley DB 独自の関数名は db_ で始まる

Berkeley DB 2.1 (1997)

- Sleepycat Software からリリース
  Sleepycat は Berkeley DB の著者らが Berkeley DB の商用サポートのために作った会社
  (2006年に Oracle が買収)
- ndbm.h は提供されなくなった
- ヘッダは以下のように使う
  #define DB_DBM_HSEARCH 1
  #include <db.h>
- ライブラリは libdb

qdbm (2003)

- 独自の API もあり高機能 (B木など)
- ndbm 互換 API を含む
- ファイルは .pag と .dir のふたつ
- ndbm 互換のヘッダは relic.h
- ライブラリは libqdbm

ライブラリの正しい使い方

対応するヘッダとライブラリを使う

- 適切なヘッダを include する
- 適切なライブラリをリンクする

正しく使うのは簡単でない

- ndbm.h は名前から素性がわからない
- libc に ndbm 互換ライブラリが入っているかもしれない
  - 4.3BSD ndbm が入っているかも
  - Berkeley DB が入っているかも
  - 入っていないかも
- libndbm は名前から素性がわからない

Ruby で起きた問題の例 (1)

- 1997-10-29 [ruby-list:5168] 青山和光
  「RedHat の標準環境では dbm がうまくコンパイルされないようです。
  require "dbm" で、ruby: can't resolve symbol 'dbm_clearerr' となります。」
- バージョンは明示されていないが、
  Ruby 1.0-971021 のリリース 1週間後

Ruby で起きた問題の例 (2)

- 1999-08-16 [ruby-list:16167] 大原重樹
  「BSD/OS 3.1 上で Ruby 1.4.0 を make しようとすると、以下のように、ext/dbm の make でエラーになってしまいます。
  dbmopen.o: Definition of symbol _dbm_open (multiply defined)
  # Ruby 1.3.6 からこうなっていました。」

何が起きていたのか

- ruby: can't resolve symbol 'dbm_clearerr'
  Berkeley DB 1 のヘッダと
  libgdbm を組み合わせてしまった
- _dbm_open (multiply defined)
  Berkeley DB 1 のヘッダと
  libgdbm と
  Berkeley DB 1 が入っている libc を組み合わせてしまった

can't resolve symbol 'dbm_clearerr'
当時の ext/dbm の状況

- Ruby 1.0-971021 の ext/dbm/extconf.rb

```
$LDFLAGS = "-L/usr/local/lib"
have_library("gdbm", "dbm_open") or
  have_library("dbm", "dbm_open")
if have_func("dbm_open")
  create_makefile("dbm")
end
```

- まず libgdbm を試し、なければ libdbm を試す
- ヘッダは調べない: ndbm.h 固定
- dbm_clearerr の存在は確認しない
- dbm.c では dbm_clearerr を単に呼び出している

can't resolve symbol 'dbm_clearerr' ライブラリの状況

ライブラリ側の dbm_clearerr の宣言と定義

- Berkeley DB 1
  - ndbm.h: 宣言なし ←
  - libdb: 定義あり (ndbm.o)
- gdbm
  - ndbm.h: #define dbm_clearerr(dbf)
  - libgdbm: 定義なし ←
- Berkeley DB は関数、gdbm は空マクロ

can't resolve symbol 'dbm_clearerr' 問題が発生する状況

- dbm_clearerr の存在を検査せずに使っていた (Ruby 1.1b9_09 まで)
- Berkeley DB 1 ndbm.h: dbm_clearerr 宣言なし
- libgdbm: dbm_clearerr 定義なし

- dbm_clearerr を使っているのに定義がない

can't resolve symbol 'dbm_clearerr' 修正？

- Ruby 1.1b9_10 で dbm_clearerr は検査されるようになった
  extconf.rb:
  ```
  have_func("dbm_clearerr")
  ```
  dbm.c:
  ```
  #ifdef HAVE_DBM_CLAERERR
        dbm_clearerr(dbm);
  #endif
  ```
  しかし typo してる
- ともあれ dbm_clearerr は使われなくなったのでリンクエラーは出なくなった

追記

- typo は Ruby 1.3.6 で修正された
- have_library("dbm", "dbm_open")
  libdbm は ndbm 互換ライブラリじゃない

- 正しい修正ではない
  Berkeley DB と gdbm は両方とも
  dbm_clearerr を提供している
  原因はヘッダとライブラリが合っていない点

_dbm_open (multiply defined) 発症

- Ruby 1.3.6 から発生
- HAVE_DBM_CLAERERR の typo 修正がきっかけ
- BSD/OS 3.1
  4.4BSD なので libc に Berkeley DB 1 が入っている

_dbm_open (multiply defined)
dbm_openの状況

ライブラリ側の dbm_open の宣言と定義

- Berkeley DB 1
  - ndbm.h: DBM　　*dbm_open();
  - libc: 定義あり (ndbm.o)　　←
- gdbm
  - ndbm.h: DBM　　*dbm_open();
  - libgdbm: 定義あり (dbmopen.o)　　←

_dbm_open (multiply defined)
dbm_clearerrの状況

ライブラリ側の dbm_clearerr の宣言と定義

- Berkeley DB 1
  - ndbm.h: 宣言なし (たぶん) ←
  - libc: 定義あり (ndbm.o) ←
- gdbm
  - ndbm.h: #define dbm_clearerr(dbf)
  - libgdbm: 定義なし ←

_dbm_open (multiply defined) 問題が発生する状況

- プログラムが dbm_open と dbm_clearerr をどちらも使っている
- Berkeley DB 1 のヘッダを使う
  dbm_open と dbm_clearerr は関数
- libgdbm をリンクする
- libc に Berkeley DB 1 の ndbm.o が入っている
- リンクはオブジェクトファイル単位に行われる

_dbm_open (multiply defined) 問題が発生する流れ

- dbm_open を使っている
  → dbmopen.o(libgdbm) がリンクされる
- dbm_clearerr を使っている
  → ndbm.o(libc) がリンクされる
- dbmopen.o と ndbm.o の両方に dbm_open が入っている
  → エラー

_dbm_open (multiply defined) 修正?

- [ruby-list:16169] eban
  1. gdbm はリンクしない
  2. GDBM の ndbm.h を include する
  という解決策が考えられます. (中略)

  gdbm がリンクされるときは dbm_clearerr のテストをしないってのが簡単かな.
- [ruby-list:16170] matz
  |gdbm がリンクされるときは dbm_clearerr の
  |テストをしないってのが簡単かな.
  そうするようにします。1.4.1からは。

_dbm_open (multiply defined) テストしないとなぜ直るのか

- Berkeley DB 1 の ndbm.h と libgdbm を使う
- gdbm なので dbm_clearerr はテストしない
- dbm_clearerr を呼び出さない
- ndbm.o がリンクされない
- multiply defined にならない

_dbm_open (multiply defined) それで問題は起きないのか

- gdbm の dbm_clearerr は空マクロ
  呼び出さなくても挙動は変わらない
- datum 構造体は一致
  - 順番も一致
  - dsize は両方 int (SUSv2 では size_t だけど)
- DBM_INSERT などの定数値も一致
- 使っている範囲では関数宣言も一致

どうも問題は起きない模様

ヘッダには違う点もある

- 使ってないけど dbm_pagfno は異なる
  .pag ファイルをオープンした fd を返す関数
  標準化されていないが、4.3BSD から存在する
  - Berkeley DB 1
    ```
    #define dbm_pagfno(a)  \
      DBM_PAGFNO_NOT_AVAILABLE
    ```
  - gdbm
    ```
    extern int       dbm_pagfno ();
    ```
- DBM 構造体の定義も異なる
  ポインタしか使わないから影響しないけど

dbm_pagfno を使いたい

- すべての fd に close-on-exec flag をつけたい
  - 理想: dbm_open に O_CLOEXEC を渡すとそうなる
  - 現実: 必ずしもそうはならない (gdbm 1.10未満とか)
  - 対処: DBM 構造体から fd を取り出して設定
- fd は dbm_pagfno と dbm_dirfno で取り出す
- Berkeley DB には dbm_pagfno はない
  データベースファイルはひとつだから
- gdbm には両方ある
- dbm_pagfno と dbm_dirfno を正しく確認するには
  対応するヘッダを使う必要がある

現在、よくある状況

- gdbm (Debian)
- Berkeley DB (RedHat)
- libc 内に Berkeley DB (4.4BSD)
- libc 内に ndbm (System V)

現在、珍しくない状況

- libc 内に ndbm があり、
  加えて gdbm か Berkeley DB をインストール
  ndbm のサイズ制限を避ける人は多い
- gdbm と Berkeley DB を両方インストール
  パッケージで簡単にインストールできるし

組み合わせ空間
header * library * libc

以下の組み合わせを考える

- 利用するヘッダ
- 明示的にリンクするライブラリ
- libc に含まれているライブラリ

ヘッダとライブラリの対応の検査

- あるヘッダとライブラリの組み合わせで、規格の範囲内コードがリンクエラーになったらその組み合わせは間違い
  例: dbm_clearerr() がリンクできる
- 各ライブラリの内部実装
  例: ヘッダが _GDBM_H_ を定義したらgdbm
- クロスコンパイルを考慮し、実行して試すのは避けたい

内部実装のバージョン依存性

- ライブラリの内部実装はバージョンによって異なる
- ヘッダもバージョンによって異なる
  例: gdbm の ndbm.h が _GDBM_H_ を定義するのは gdbm 1.9.1 以降
  _GDBM_H_ を定義するのは gdbm.h だが、gdbm 1.8.3 以前は gdbm.h を include していない

各 OS による派生版もある

- Berkeley DB の ndbm.h は Net/2 の時代から db.h を include しており、db.h は _DB_H_ を定義する
- しかし、Mac OS X の ndbm.h は db.h を include せず、_DB_H_ を定義しない
- Mac OS X での変更
  FreeBSD で変えた形跡はない

組み合わせで考慮しないもの

- qdbm
  ndbm.h や libc として提供される例がなく、混乱が起きにくい
- libndbm
  古い OS で libndbm を提供している例があるが、dbm/extconf.rb はデフォルトでは探さない

ヘッダの分類

- ndbmh　　ndbm
- db1h　　Berkeley DB 1
- db2h　　Berkeley DB 2 以降
- g18h　　gdbm 1.8.3以前
- g19h　　gdbm 1.9以降

ライブラリの分類

- libc　　libc に入っているものを使う
- db1　　libdb をリンク　　Berkeley DB 1
- db2　　libdb をリンク　　Berkeley DB 2以降
- g17　　libgdbm をリンク　GDBM 1.8.0以前
- g18　　libgdbm をリンク　GDBM 1.8.1〜1.8.3 (ndbm関数なし)
- g19　　libgdbm をリンク　GDBM 1.9以降 (ndbm関数なし)
- g18c　libgdbm_compat, libgdbm をリンク　GDBM 1.8.1〜1.8.3
- g19c　libgdbm_compat, libgdbm をリンク　GDBM 1.9以降

libc の分類

- ndbm-libc　ndbm が入っている
- db1-libc　Berkeley DB 1 が入っている
- glibc　入っていない
  (uClibc など、glibc 以外も含む)

組み合わせ空間

- ヘッダ　　5種類
- ライブラリ　8種類
- libc　　3種類
- 計　5×8×3 = 120種類

組み合わせ空間の表

		libc	db1	db2	g17	g18	g19	g18c	g19c
ndbm-libc	ndbmh								
	db1h								
	db2h								
	g18h								
	g19h								
db1-libc	ndbmh								
	db1h								
	db2h								
	g18h								
	g19h								
glibc	ndbmh								
	db1h								
	db2h								
	g18h								
	g19h								

正しい組み合わせ

		libc	db1	db2	g17	g18	g19	g18c	g19c
ndbm-libc	ndbmh	■							
	db1h		■						
	db2h			■					
	g18h				■			■	
	g19h								■
db1-libc	ndbmh								
	db1h	■	■						
	db2h			■					
	g18h				■			■	
	g19h								■
glibc	ndbmh								
	db1h		■						
	db2h			■					
	g18h				■			■	
	g19h								■

dbm_open がリンクできる (1)

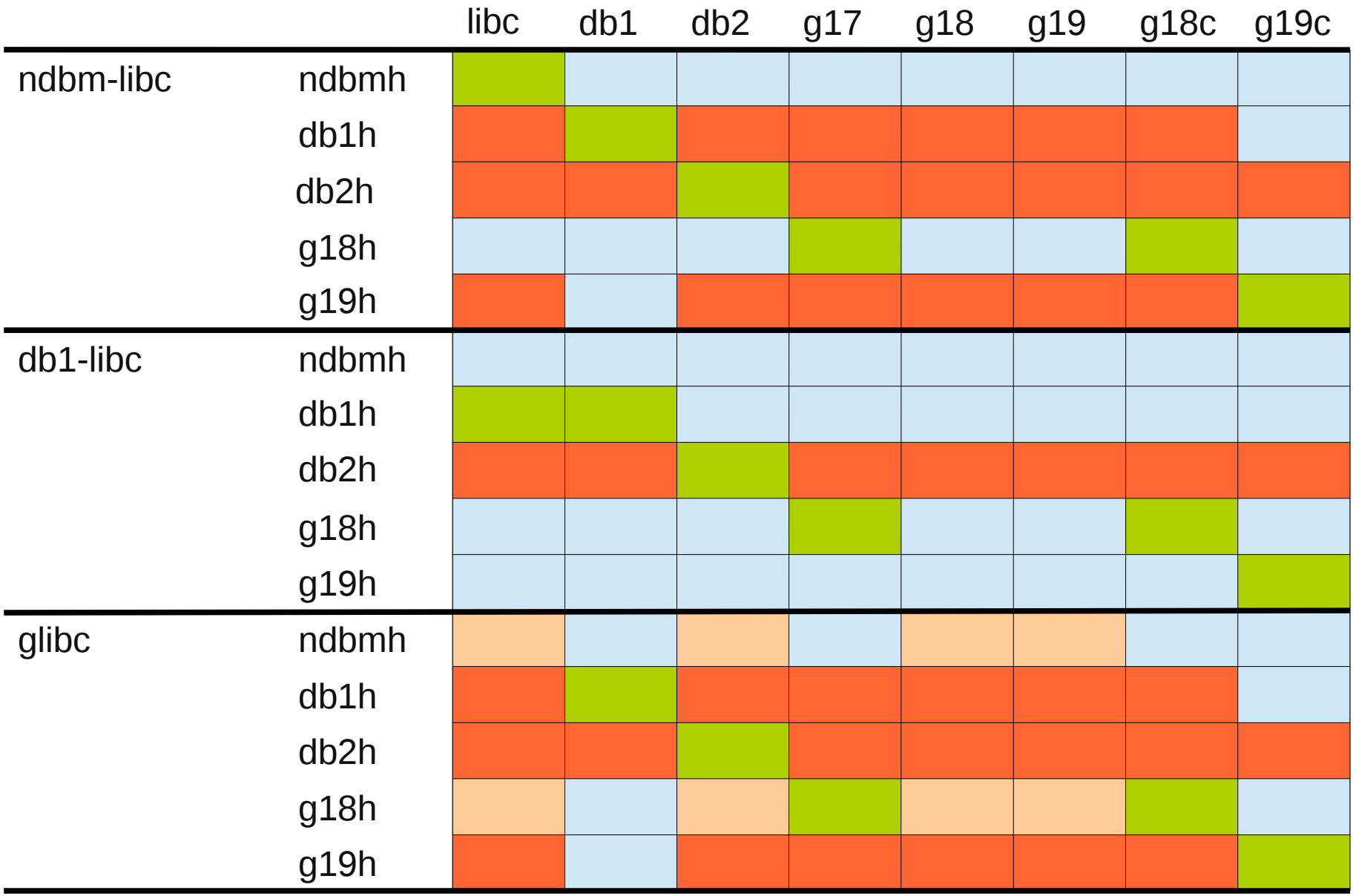
- ヘッダ
  ndbmh: DBM *dbm_open();
  db1h: DBM *dbm_open(const char *, int, int);
  db2h: #define dbm_open(a, b, c) \
  __db_ndbm_open(a, b, c)
  g18h: DBM *dbm_open();
  g19h: DBM *dbm_open(char *, int, int);
- Berkeley DB 2 以降はライブラリに
  dbm_open 関数は存在しない
- __db_ndbm_open は db2 にのみ存在する

dbm_open がリンクできる (2)

dbm_clearerr がリンクできる (1)

- ndbmh: #define dbm_clearerr(db) \
  ((db)->dbm_flags &= ~_DBM_IOERR)
  db1h: なし
  db2h: #define dbm_clearerr(a) \
  __db_ndbm_clearerr(a)
  g18h: #define dbm_clearerr(dbf)
  g19h: void dbm_clearerr (DBM *dbf);
- ndbm, db2, g1[789], g18c はライブラリ内に dbm_clearerr は存在しない
- __db_ndbm_clearerr は db2 にのみ存在する

dbm_clearerr がリンクできる (2)



_GDBM_H_ があるなら libgdbm_compat 一択 (1)

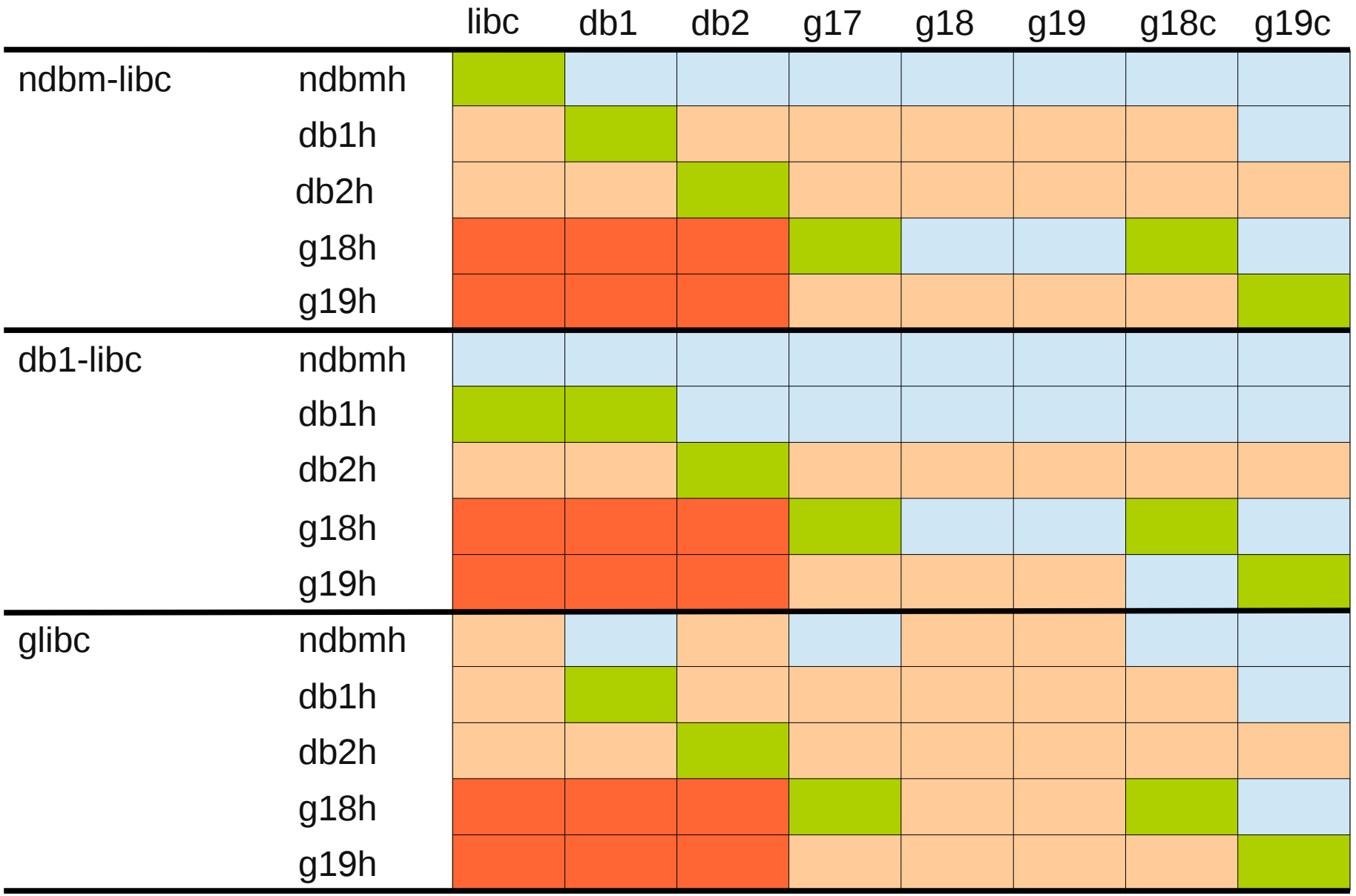
- _GDBM_H_ を定義するか
  ndbmh: しない
  db1h: しない
  db2h: しない
  g18h: しない (gdbm.h を include しない)
  g19h: する (gdbm.h を include する)
- g1[89] は libgdbm に ndbm 互換関数がない

_GDBM_H_ があるなら libgdbm_compat 一択 (2)

gdbmならgdbm_version が存在 (1)

- gdbm のヘッダ検出
  - g19h: _GDBM_H_ の存在確認
  - g18h: dbm_clearerr が空マクロか?
    dbm_clearerr(foobarbaz) はコンパイルできるか?
- libgdbm は常に gdbm_version を定義する

gdbmならgdbm_version が存在 (2)

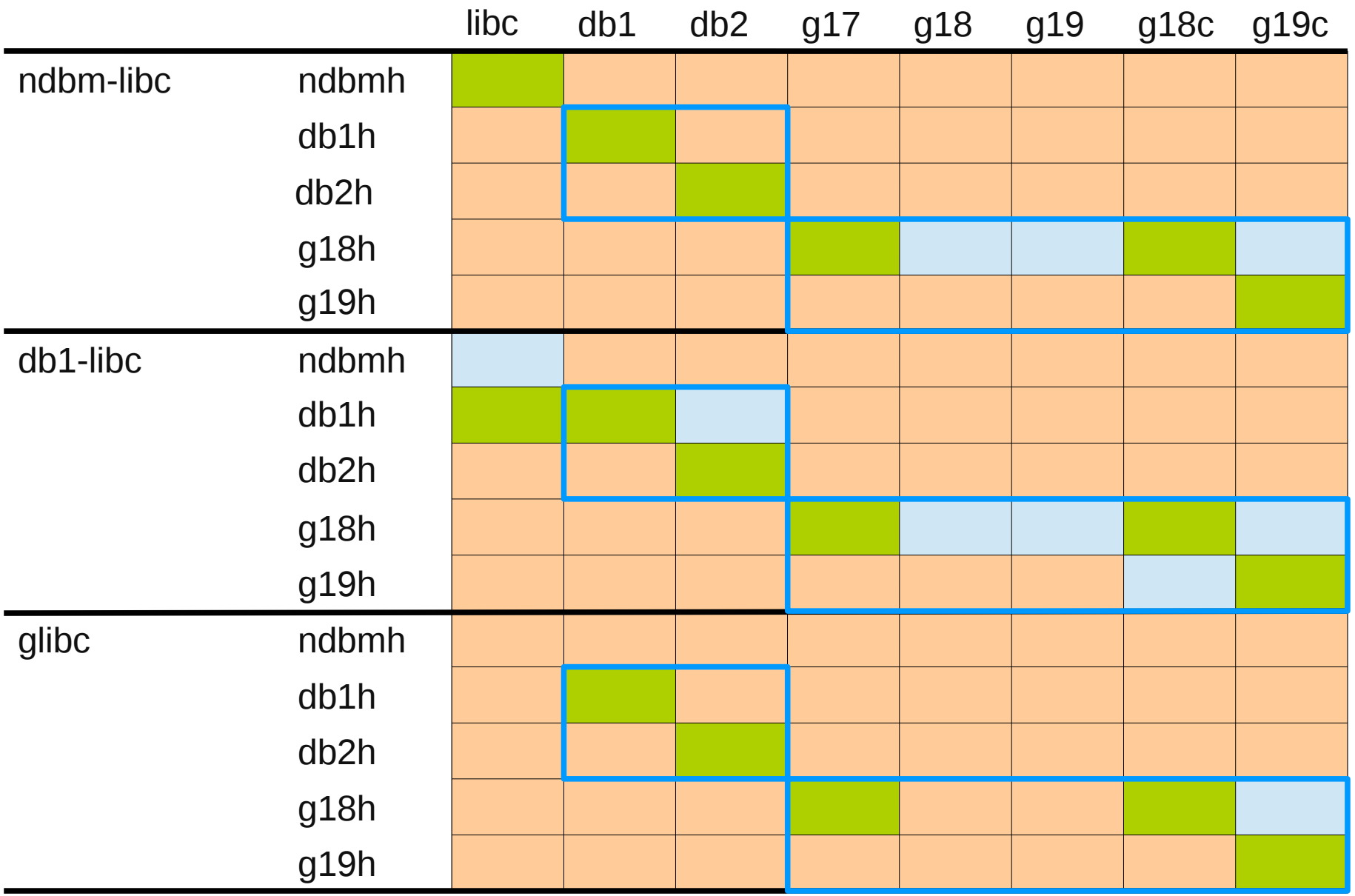


ヘッダの中身とライブラリ名 (1)

- ヘッダ名が ndbm.h でも素性は調べられる
  - g18h, g19h: 前述のとおり
  - ndbmh: _DBM_IOERR マクロを定義する
    #define _DBM_IOERR　0x2
  - db1h, db2h: DBM_SUFFIX マクロを定義する
    #define DBM_SUFFIX　　".db"
    (_DB_H_ は Mac OS X のために避ける)
- libc, libndbm以外のライブラリは名前で素性がわかる (バージョンはわからないかもしれない)
- 不適切な組み合せを検出

ヘッダの中身とライブラリ名 (2)

現状


libc
db1
db2
g17
g18
g19
g18c
g19c
ndbm-libc
ndbmh
db1h
db2h
g18h
g19h
db1-libc
ndbmh
db1h
db2h
g18h
g19h
glibc
ndbmh
db1h
db2h
g18h
g19h

現状

- 多くの状況では正しく検出するが、完璧ではない
- 検出に失敗する状況
  - 複数のバージョンの Berkeley DB
    複数のバージョンの gdbm
    普通ヘッダとライブラリは同時にインストールされるので、それらのバージョン違いはなかなか起きない
  - 4.4BSD に 4.3BSD ndbm をインストールした場合
    libc 内の Berkeley DB 1 と 4.3 ndbm の ndbm.h の組み合わせ

    4.3BSD ndbm は単独では配布されていないので 4.4BSD にインストールすることはまずない

libndbm の混乱 (1)

- glibc
  - 1997: glibc 2.0 は Berkeley DB 1.85 を同梱
    ヘッダは ndbm.h、ライブラリは libdb
  - 1999: glibc 2.1 は Berkeley DB 2 を追加
    Berkeley DB 1: ndbm.h, libdb1
    Berkeley DB 2: db.h, libndbm は libdb へのリンク
  - 2000: glibc 2.2 で両方外した
    Berkeley DB の変化が激しすぎるため
  - Mandriva Linux は libdb2-devel パッケージで
    libndbm を提供 (互換性のため?)

libndbm の混乱 (2)

- OpenSuSE の gdbm-devel パッケージは ndbm.h と libndbm を提供
- Tru64 UNIX は Berkeley DB の ndbm を libndbm として提供
- SCO OpenServer, Unixware は 4.3BSD ndbm を libndbm として提供

歴史的な OS を考慮すれば、libndbm は ndbm, gdbm, Berkeley DB のすべての可能性がある
Ruby では libndbm はデフォルトでは探さないが、オプションで指定可能 (未テスト)

dbm に関連した他の話題

- ファイル形式が違う
  - ライブラリによって拡張子が違う: pag, dir, db
  - ファイルの数も違う
  - 中身の形式も違う
- 複数のライブラリを同時にリンクできないことがある

複数のライブラリを同時にリンク

- Berkeley DB dbm_open in libc is called even if --with-dbm-type=gdbm_compat is specified on FreeBSD [ruby-dev:45262]
  同じシンボルを提供している複数のライブラリをリンクすると片方しか参照できない
- Berkeley DB 2 以降はマクロで名前を置き換えているので問題が起きない
- gdbm は libgdbm_compat に分離したので使わなくても済む
- qdbm は libqdbm に入れている

ndbm 互換ライブラリ作者への要望

- ndbm.h を提供するなら、XXXDBM_NDBM_H など判断しやすいマクロ定義してください
- dbm_open などはマクロにして、ほかのライブラリとシンボルが衝突しないようにしてください
- 十分に独特な名前をつけてください
  db の 2文字だけというのは一般的すぎます

まとめ

- Ruby の dbm 拡張ライブラリがさまざまな ndbm 互換ライブラリに対応する仕掛けを述べた
- 多くの環境では正しく動くようになっている
- C で互換ライブラリを作る人に要望を述べた

多様性と戦え

dbm やブラウザ以外にも多様性はある

- OS は常に増えている
  - libc
  - kernel
- 最近 Ruby 処理系はたくさんあるようだ