Concord: プロトタイプ方式の オブジェクト指向データベースの試み

守岡 知彦

1 はじめに

現在、プログラム開発におけるリファクタリングの重要性は広く認識されるようになったといえる。しかしながら、データベースの場合、プログラム程リファクタリングは盛んではない。これは1つにはデータベースの場合、リファクタリングが容易ではないからだと考えられる。

現在普及しているデータベース・システムにはリレーショナル・データベース、全文データベース、XML データベースなどがある。また、この他、オブジェクト指向データベースや知識データベースといったものもある。これらは大別すると、スキーマやクラス定義といったデータ形式の定義をあらかじめ行う必要があるものと、そうでないものに分けることができる。リレーショナル・データベースや一般的なオブジェクト指向データベースは前者であり、全文データベースは後者である。XML データベースの場合、DTD やスキーマ定義を要するものとそうでないものがあり、両方の性質を持つものもあり得る。

データ形式の定義をあらかじめ行うタイプのデータベース・システムではデータ形式の構造を活かしたデータ操作や入力データのチェックが可能であるが、当然のことながら、予めデータ形式が明確に定義されている必要がある。そして、一度データベースを運用し始めると、途中でデータ形式を変更するのは面倒であることが多い。また、異なるデータ形式のデータベースを混在させるのも面倒である。

これに対し、データ形式の定義を必要としないタイプのデータベース・システムにはこの種の面倒が少ない。しかしながら、データの構造を活かしたチェックや操作も基本的に行えない。

複雑な対象やレガシーシステムを扱う場合、最初に多大な労力をかけて精密にデータベースのスキーマを設計するよりも、リファクタリングによって、データベースのメンテナンス性を向上させ、少ない労力で品質を維持して行く方が良いといえる。このためには、対象データの構造を活かしたデータ操作や検証を可能にしつつ、データ形式の変更や複数のデータ形式の混在が容易なデータベース・システムを実現する必要がある。

著者は中国学関連のレガシーな文献データベース・システムのリニューアル作業を行ったり、唐代知識データベースの開発に触れる中で、最初にデータやその背景にある知識・体系の全貌を把握することが困難な対象の場合、データベースを容易にリファクタリングできることが重要であるという感を強く持つに至った。

また、一方で、著者らは文字を文字符号に依存せずに文字オントロジに基づいて処理するテキスト処理環境 **CHISE** [1] の開発を進めているが、ここでの文字表現・処理手法である **Chaon** モデル[3] や文字データベースの枠組と同様の方法でこうした問題を扱えるのではないかと思うに至った。

こうした観点から、著者は **Concord** というデータベース・システムの開発をはじめた。これは一種のプロトタイプベースのオブジェクト指向データベース・システムであり、実装的には CHISE の文字データベース・システムを一般化したものである。本論文ではこのシステムの概要について説明する。

2 データモデル

Concord では、オブジェクトをノードとした有向グラフで情報を表現する。オブジェクトは素性（属性）の集合で表現される。ここで、素性は名前（素性名）と値（素性値）の対である。

2.1 素性の種類

素性は大別すると

基礎素性 数値や識別子（シンボル）といったアトミックなデータ、または、それらのリストや配列を値として取る素性

関係素性 オブジェクトの集合を値として取り、値に取った各オブジェクトと素性を持つオブジェクトの関係を表す素性（オブジェクトをノードとした有向グラフのリンクとなる）

構造素性 オブジェクトを要素として持つリストや配列を値として取る素性

ID 素性 オブジェクトに対する ID（素性名においてユニークな数値または識別子）を値として取る素性。ある ID 素性において、その素性を持つ各オブジェクトとその素性値である ID は 1 対 1 対応していなければならない（これにより、値である ID をキーにオブジェクトを得るための索引を作ることができる）

写像素性 オブジェクトに対応する ID を値として取る素性。素性値に対応するオブジェクトが複数あっても良い（ID 素性は写像素性の特殊な形と看做すことができる）

メタデータ素性 元になる素性に対するメタデータを記述するための素性

に分類することができる。

2.2 類型

Concord には本質的な意味でのクラスは存在しない。しかしながら、あらゆるオブジェクトを単一の空間に置くのは不合理であることも少なくないので、オブジェクトの種類毎に異なる空間を設定することが可能である。これを**類型 (genre)** と呼ぶ。

3 libconcord

Concord の基本部分は `libconcord` によって実現されている。これは C で書かれたライブラリであり、C の API を提供する。API 上、データベースのバックエンドを拡張することが考慮されているが、現在のところ Berkeley DB を用いた実装のみが実現されている。

3.1 API

3.1.1 データスイート (DS)

データスイート (data suite; DS) は 1 つ以上の類型 (genre) からなるひとかたまりのデータベースのことである。

DS はそれが置かれる場所（ファイル名）[1] によって管理される。

型 **CONCORD_DS**
: DS (data suite) を表す型。

CONCORD_DS **concord_open_ds** (CONCORD_Backend_Type *type,* const char* *location,* int *subtype,* int *modemask*)
: 場所 *location* に存在する DS を開き、それに対応する CONCORD_DS 型のオブジェクト（DS オブジェクト）を返す。
: 引数 *type* はバックエンドの種類を表す。現在の所 CONCORD_Backend_Berkeley_DB のみが指定可能である。
: 引数 *location* は DS の場所を指す。
: 引数 *subtype* は DS 中の各素性の表現方法の種類の既定値を表す。その解釈は *type* に依存する。省略したい場合は 0 を指定する。
: *modemask* は DS 中の各素性の情報を格納するファイルのパーミッションの既定値を表す。
: 処理が失敗した場合、NULL が返る。

int **concord_close_ds** (CONCORD_DS *ds*)
: 引数 *ds* で指定された DS を閉じる。
: 処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

char* **concord_ds_location** (CONCORD_DS *ds*)
: 引数 *ds* で指定された DS の場所を返す。

3.1.2 類型

型 **CONCORD_Genre**
: 類型 (genre) を表す型。

CONCORD_Genre **concord_ds_get_genre** (CONCORD_DS *ds,* const char* *name*)
: 引数 *ds* で指定された DS において、文字列 *name* を名前として持つ類型を開き、それに対応する CONCORD_Genre 型のオブジェクト（類型オブジェクト）を返す。
: 処理が失敗した場合は NULL を返す。

char* **concord_genre_get_name** (CONCORD_Genre *genre*)
: 類型 *genre* の名前を返す。

CONCORD_DS **concord_genre_get_data_source** (CONCORD_Genre *genre*)
: 類型 *genre* の DS を返す。

int **concord_genre_foreach_feature_name** (CONCORD_Genre *genre,* int (**func*) (CONCORD_Genre *genre,* char* *name*))
: 類型 *genre* 内に存在する各素性に対して関数 *func* を呼び出す。この時、関数 *func* の第 1 引数には類型、第 2 引数には素性名が渡される。
: 関数 *func* が 0 以外の値を返すと、その時点で処理は終了する。

[1] 将来、分散化される場合には URI を用いることになるだろう。

3.1.3 素性

型 **CONCORD_Feature**
　素性を表す型。

CONCORD_Feature **concord_genre_get_feature** (CONCORD_Genre *genre,* const char* *name*)
　類型 *genre* において、文字列 *name* を名前として持つ素性を開き、それに対応する CONCORD_Feature 型のオブジェクト（素性オブジェクト）を返す。
　処理が失敗した場合は NULL を返す。

char* **concord_feature_get_name** (CONCORD_Feature *feature*)
　素性 *feature* の名前を返す。

CONCORD_Genre **concord_feature_get_genre** (CONCORD_Feature *feature*)
　素性 *feature* が属する類型を返す。

int **concord_feature_setup_db** (CONCORD_Feature *feature,* int *writable*)
　素性 *feature* に対応するバックエンド・ストレージを開く。
　処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。
　引数 *writable* に 0 以外が指定された場合、書き込み可能なモードでバックエンド・ストレージを開く。この時の引数をチェックすることにより、素性 *feature* に対して値を書き込むことが可能であるかどうかを調べることができる。

int **concord_feature_sync** (CONCORD_Feature *feature*)
　素性 *feature* に関する（書き込み用）キャッシュをストレージ上に書き出す。
　処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

int **concord_obj_put_feature_value_str** (const char* *object_id,* CONCORD_Feature *feature,* unsigned char* *value*)
　文字列 *object_id* を ID とするオブジェクトの素性 *feature* に値 *value* を設定する。
　処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

int **concord_obj_get_feature_value_string** (const char* *object_id,* CONCORD_Feature *feature,* CONCORD_String *value*)
　文字列 *object_id* を ID とするオブジェクトの素性 *feature* の値を *value* に格納する。
　処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

unsigned char* **concord_obj_gets_feature_value** (const char* *object_id,* CONCORD_Feature *feature,* unsigned char* *dst,* size_t *size*)
　文字列 *object_id* を ID とするオブジェクトの素性 *feature* の値を *dst* で指される場所に格納するとともに、C の文字列として返す。
　size は *dst* で指される記憶領域の大きさを示す。
　処理が失敗した場合は NULL が返る。

int **concord_feature_foreach_obj_string** (CONCORD_Feature *feature,* int (**func*) (CONCORD_String *object_id,* CONCORD_Feature *feature,* CONCORD_String *valdatum*))
　素性 *feature* を持つ各オブジェクトに対し、関数 *func* を呼び出す。この時、その関数の第 1 引数にはオブジェクト ID, 第 2 引数には素性、第 3 引数には素性値が渡される。
　関数 *func* が 0 以外の値を返すと、その時点で処理は終了する。

3.1.4 索引

索引 **(index)** とは、ID 素性の値をキーにして対応するオブジェクトを得るための表のことである。

型 **CONCORD_INDEX**
　索引を表す型。

CONCORD_INDEX **concord_genre_get_index** (CONCORD_Genre *genre,* const char* *name*)
類型 *genre* において、文字列 *name* を名前として持つ索引を開き、それに対応する CONCORD_INDEX 型のオブジェクト（索引オブジェクト）を返す。
処理が失敗した場合は NULL を返す。

int **concord_index_setup_db** (CONCORD_INDEX *index,* int *writable*)
索引 *index* に対応するバックエンド・ストレージを開く。
処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。
引数 *writable* に 0 以外が指定された場合、書き込み可能なモードでバックエンド・ストレージを開く。この時の引数をチェックすることにより、索引 *index* に対して ID 素性値を書き込むことが可能であるかどうかを調べることができる。

int **concord_index_sync** (CONCORD_INDEX *index*)
索引 *index* に関する（書き込み用）キャッシュをストレージ上に書き出す。
処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

int **concord_index_strid_put_obj** (CONCORD_INDEX *index,* const char* *strid,* char* *object_id*)
索引 *index* において ID 素性値 *strid* に対応するオブジェクトの ID *object_id* を設定する。
処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

int **concord_index_strid_get_obj_string** (CONCORD_INDEX *index,* const char* *strid,* CONCORD_String *object_id*)
索引 *index* において ID 素性値 *strid* に対応するオブジェクトの ID を *object_id* に格納する。
処理が成功した場合は 0 を返し、それ以外の場合には 0 以外の値が返る。

3.1.5 その他

型 **CONCORD_String_Tank**
結果の文字列を格納するための構造体。

型 **CONCORD_String**
CONCORD_String_Tank のポインタ。

int **CONCORD_String_size** (const CONCORD_String *s*)
文字列 *s* の長さを返す。

unsigned char* **CONCORD_String_data** (const CONCORD_String *s*)
文字列 *s* に格納されたデータを返す。

3.2 データ表現

libconcord は現在のところ Berkeley DB を用いたバックエンドのみが存在する。
Berkeley DB を用いたバックエンドでは

location/*genre*/`feature`/*feature*

というディレクトリ構造でデータスイート (DS) を表現し、1 素性を 1 つの Berkeley DB のファイルに格納するようになっている。

ここで、*location* は DS の場所であり、*genre, feature* は、それぞれ、類型、素性の名前を元に生成したものである。

この生成方式は、名前を UTF-8 で符号化した上で、その中に `% / \ : * ? " < > |` が含まれていた場合、URL エンコーディングと同様な %XX （但し、XX は ASCII code を表す 16 進数）で表すというものである。

これにより、ある DS のある類型のある素性が固有の 1 ファイルに割り当てられる。この Berkeley DB ファイルはキーがオブジェクト ID, 値が素性値となっている。

Concord では素性の素性を設定することが可能であるがこの場合、

location/`feature`/`feature`/素性の素性

に素性の素性が格納される。ここで、この Berkeley DB ファイルのキーは素性名である。すなわち、Concord では素性もまた `feature` という類型に属するオブジェクトの一種であり、素性の定義は同一 DS 中では共通であるということがいえる。

索引は同様に

location/*genre*/`index`/*feature*

に格納される。ここで、*feature* は索引の名前を元に生成したものである。また、この Berkeley DB ファイルのキーはこの索引に対応する ID 素性の値であり、値はオブジェクトである。

Berkeley DB ファイル中のキーとなるオブジェクト ID および素性値は現在のところ Emacs Lisp の S 式を UTF-8 化したもので表現することにしている。[2] この S 式では Concord のオブジェクトは

#s(concord-object genre *genre* =id *id*)

のように表現される。ここで、*genre* は類型名、*id* はオブジェクト ID である。

4 XEmacs CHISE

Concord の C 以外の言語処理系でのバインディングとしては今のところ XEmacs CHISE によるものが存在する。これは XEmacs CHISE 上において Concord の Emacs Lisp API を提供するものであり、この機能はコンパイル時のオプションで選択することができる（libconcord がインストールされていれば、configure はデフォルトでこの機能の利用を選択する）。

4.1 Emacs Lisp API

XEmacs CHISE が提供する Concord API について説明する。この Emacs Lisp 版 API の方が C 版の API よりも構造データを扱う上では直観的である。この意味では C 版が低レベル API を提供しているのに対し、Emacs Lisp 版は高レベル API を提供しているといえる。

機能 **concord**

Concord 機能を提供している場合、この機能が定義される。

［例］(featurep 'concord)
;; Concord 機能が利用可能な場合、`t` が返る。

関数 **concord-assign-genre** (*genre directory*)

類型 *genre* に対して *directory* で指定された場所に存在する DS を割り当てる。

［例］(concord-assign-genre 'work "/usr/local/var/ruimoku/db")

[2] Scheme の S 式を採用することも検討している。

関数 **concord-genre-directory** (*genre*)

類型 *genre* に割り当てられたディレクトリを返す。

関数 **concord-make-object** (*genre* ***&optional*** *id ds*)

類型 *genre* に属する concord-object を作成し、それを返す。

省略可能な引数 *id* が指定された場合、これをオブジェクト ID とする（省略しないことを推奨する)。

関数 **concord-object-p** (*object*)

引数 *object* で指定されたものが concord-object である場合、t を返す。

［例］ (concord-object-p 1)
→ nil

(concord-object-p (concord-make-object 'work 'B021133))
→ t

関数 **concord-object-id** (*object*)

Concord-object *object* のオブジェクト ID を返す。

関数 **concord-object-get** (*object feature*)

Concord-object *object* の素性 *feature* の値を返す。

［例］ (concord-object-get #s(concord-object genre work =id B021133) '=id)
→ B021133

(concord-object-get #s(concord-object genre work =id B021133) 'publication/place)
→ nil

関数 **concord-object-put** (*object feature value*)

Concord-object *object* の素性 *feature* に値 *value* を設定する。

素性 *feature* が ID 素性（素性名の先頭が = ではじまり、その次が > でないもの）の場合、その ID 素性に対応する索引の *value* の項目に *object* が自動的に登録される。

［例］ (concord-object-put #s(concord-object genre work =id B021133)
'publication/place "上海")
→ t

(concord-object-get #s(concord-object genre work =id B021133)
'publication/place)
→ "上海"

関数 **concord-object-spec** (*object*)

Concord-object *object* の持つ素性の集合を連想リストとして返す。

［例］ (concord-object-spec #s(concord-object genre work =id B021133))
→ ((writing-system . cjk)
(title . "曾国藩的幕僚们")
(source . "wachuto")
(publication/year . 2000)

(publication/publisher . "东方出版中心")
(publication/place . "上海")
(publication/month . 10)
(paper-size . "21cm")
(page . 449)
(ncid . BA52855639)
(item-type . book)
(genre/code . "013X")
(creators/au #s(concord-object genre person =id B021133/1))
(content/type/code . 1)
(content/region1/code . 110000)
(content/period/oldest/year . 1851)
(content/period/older/modifier . J)
(content/period/older/century . 19)
(content/period/newest/year . 1900)
(content/code . "11D+19JXX1")
(=id . B021133))

関数 **concord-feature-list** (*genre* ***&optional*** *ds*)

類型 *genre* に存在する素性名の集合をリストとして返す。

［例］(concord-feature-list 'era)
→ (year/oldest year/newest region/ruimoku/code name =ruimoku-era-code =id)

関数 **concord-for-each-object-in-feature** (*function feature genre* ***&optional*** *ds*)

類型 *genre* 内に存在する素性 *feature* を持つ各オブジェクトに対し関数 *function* を適用する。

引数 *function* で指定される関数には２つの引数が適用される。その第１引数は concord-object であり、第２引数はそのオブジェクトにおける素性値である。この関数が 非-nil 値を返した時点で処理は終了される。

［例］
```
(concord-for-each-object-in-feature
 (lambda (obj val)
   (insert (format "%S\t%S\n" obj (concord-object-spec obj)))
   nil)
 '=id 'era)
```

関数 **concord-decode-object** (*feature value genre* ***&optional*** *ds*)

類型 *genre* に属する各オブジェクトの内、ID 素性 *feature* の値が *value* であるオブジェクトを索引から探索し、見つかったオブジェクトを返す。見つからなかった場合は nil を返す。

5 CHISE との関係

Concord のデータモデルは CHISE における文字表現モデルである **Chaon** モデル [3] と同じものである。但し、Chaon モデルが文字だけを対象としていたのに対し、Concord はオブジェクト一般を対象としている点が異なる。

Concord の基盤ライブラリである libconcord は libchise [2] の実装を元に、これを一般化する形で実装されている。このため、データ表現等も CHISE のものと上位互換になっている。libchise

との主な相違点は、libchise にはなかった類型 **(genre)** が導入されていることと、CCS が索引 **(index)** という機構に一般化されていることである。

また、現在の libchise の実装は libconcord を使ったものとなっており、CHISE は Concord のアプリケーションの 1 つになっている。ここで、CHISE の文字オブジェクトは `character` という類型として扱われる。CCS も索引を使って実現されている。

6 クラスベースのオブジェクト指向との関係

6.1 クラス

Concord はプロトタイプ方式のオブジェクト指向に基づいており、クラスとインスタンスの間に本質的な区別は存在せず、クラス・オブジェクトとインスタンス・オブジェクトは同一空間上に存在する。

Concord ではクラス的なものを表現したい場合、次の 2 つの方法があり得る。一つは明示的なクラス・オブジェクトを設けない場合で、もう一つは明示的なクラス・オブジェクトを設ける場合である。

前者は（求めたいクラスに属すべきインスタンス）オブジェクトの持つ素性の集合の共通部分である。

後者はクラス・オブジェクトとインスタンス・オブジェクトの関係の一種としてクラス・インスタンス関係を書く方法である。

6.2 継承

Concord でオブジェクトの継承を行いたい場合、次の 2 つの方法があり得る。一つは継承したいオブジェクトの素性の集合をコピーする方法であり、もう一つは is-a 関係を関係素性の一つとして記述する方法である。後者は差分的な記述を行うということであり、記述量を減らすのに効果的である。

CHISE ではこのための素性名として –>subsumptive と <–subsumptive, –>denotational と <–denotational という 2 対を用いているが、Concord でもこれらを用いることにしている。

7 おわりに

プロトタイプベースのオブジェクト指向データベース・システム Concord の概要について述べた。このシステムの開発はまだはじまったばかりでまだ実装面や応用面での蓄積は少ないが、CHISE Project での文字知識データベースの経験をその他の対象に活かすことが可能であるという感触を持っている。

参考文献

[1] CHISE Project. `http://kanji.zinbun.kyoto-u.ac.jp/projects/chise/`.

[2] 守岡知彦, 江渡浩一郎. 文字知識処理環境 CHISE の基盤ソフトウェア. Linux Conference 抄録集, Vol. 1, , 2003.

[3] 守岡知彦, 師茂樹. 文字素性に基づく文字処理. 情処研報, Vol. 2004, No. 58, pp. 53–60, May 2004. 2004-CH-62.