Nosql 資料庫開發-HW3

學生:潘兆新
系級:資電二
學號:111504506

此次作業實現的功能如下:

1. 資料庫架構整改
2. 雙向鏈結串列
3. 記憶體洩漏檢測

因應作業四需支援JSON資料格式，為了程式更好的擴充性，本次進行的架構調整，檔案結構如下:

db.h / db.c :

定義資料庫結構，將資料庫的功能與內部實現(HashTable)分離。

```
typedef struct DB{
    struct Dict *dict; // change to **dict if implemented cluster mode
} DB;

DB *DB_create(size_t bucket_size);
int DB_free(DB *db);
```

並且提供外部的application可使用的API，包含set, get, del, lpush, lpop, rpush等函式

```
// only available on `string`
int set(DB *db, const char * const key, const char * const val);
char* get(DB *db, const char * const key);
int del(DB *db, const char *const key);
// only available on `dlist`
int lpush(DB *db, const char * const key, const char * const val);
int lpop(DB *db, const char * const key);
int rpush(DB *db, const char * const key, const char * const val);
int rpop(DB *db, const char * const key);
int llen(DB *db, const char * const key);
int lrange(DB *db, const char * const key, int left, int right);
```

dbobj.h / dbobj.c:

模仿redisObj，建立一個general type的物件DBobj，所有型態的變數都可以由DBobj來儲存，並且統一處理變數的釋放。

```
typedef struct DBobj{
    unsigned type;
    void* value; // store low-level data type, ex: Dlist, set, tree...
} DBobj;

DBobj *DBobj_create(unsigned type);
```

```
DBobj *DBobj_create_list(DBobj *dbobj);
DBobj *DBobj_create_string(DBobj *dbobj);
void* DBobj_get_val(DBobj *dbobj);
int DBobj_free(DBobj *dbobj);
int DBobj_free_string(DBobj *dbobj);
int DBobj_free_list(DBobj *dbobj);
```

dict.h / dict.c

此檔案主要實現先前的hashtable結構(也就是redis當中實現資料庫的底層資料結構，dict)，此次經過一些修改。

struct Dict是一個hashtable，儲存許多buckets，每一個bucket都是一個單向鏈結串列(struct DictEntryList)，在collision的時候chaining。單向鏈結串列的每一個元素都是DictEntry。

dictEntry為了儲存多種形態的資料，參考redis修改成以下的結構。在union v當中，val可以儲存C原生的字串、DBobj，u64，s64則儲存其他整數類別（這種結構就不再包裝成DBobj），並且使用type來儲存該節點的資料型態 (type constant定義在其他的檔案)。

```
typedef struct DictEntry{
    char *key;
    union {
        void *val;
        uint64_t u64;
        int64_t s64;  // store unsigned / signed integer directly
    } v;
    unsigned type;
    struct DictEntry *next;
} DictEntry;

// simple list for a buckets of dict
typedef struct DictEntryList{
    DictEntry *head;
} DictEntryList;

typedef struct Dict{
    DictEntryList **bucket;
    size_t n_bucket;
    size_t n_entry;
} Dict;
```

Dict的API支援字典的新增、刪除、修改，查詢等
DictEntry的API實現單向鏈結串列的基本操作。
DictEntry的API則處理變數的content修改及變數的釋放。

dlist.h / dlist.c

dlist則是這次作業要實現的雙向鏈結串列。唯一的特點在於，每一個雙向鍊結串列的節點Dnode，都是儲存一個Dbobj的物件，這是為了將來實作其他嵌套的結構（例如JSON，可能的情況是字典當中有陣列，陣列當中還有其他的資料結構，所以使用general type的DBobj來實作。）

```
/*
  DBobj *value: create obj before calling push function
*/
typedef struct Dnode{
    DBobj *value;
    struct Dnode *prev;
    struct Dnode *next;
} Dnode;

typedef struct Dlist{
    struct Dnode *head;
    struct Dnode *tail;
    int len;
} Dlist;

Dlist* dlist_create();
int dlist_free(Dlist *list);
int dlist_push_front(Dlist *list, DBobj *val);
int dlist_pop_front(Dlist *list);
int dlist_push_tail(Dlist *list, DBobj *val);
int dlist_pop_tail(Dlist *list);
```

本次作業內容的實現：
雙向鏈結串列的操作lpush, lpop, rpush, rpop, llen, lrange的function都寫在db.c當中。
而雙向鏈結串列的實現則是在 dlist.h / dlist.c當中。

記憶體洩漏檢測：

完整的記憶體洩漏檢測，運行以下指令：

```
make
gcc -o benchmark benchmark.c -L. -ldatabase -Wl,-rpath=.
valgrind --leak-check=full --show-reachable=yes -s ./benchmark
```

下圖運行結果顯示沒有任何記憶體洩漏。

```
==1350== Memcheck, a memory error detector
==1350== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==1350== Using Valgrind-3.18.1 and LibVEX; rerun with -h for copyright info
==1350== Command: ./benchmark
==1350==
Insert  1000000 data    Total 9.735902 s        average 9.735902 us
Get     1000000 data    Total 3.800144 s        average 3.800144 us
Delete  1000000 data    Total 5.702249 s        average 5.702249 us

lpush   1000000 data    Total 9.404397 s        average 9.404397 us
rpush   1000000 data    Total 9.596898 s        average 9.596898 us
lpop    1000000 data    Total 5.868238 s        average 5.868238 us
rpop    1000000 data    Total 5.728346 s        average 5.728346 us
==1350==
==1350== HEAP SUMMARY:
==1350==     in use at exit: 0 bytes in 0 blocks
==1350==   total heap usage: 14,145,785 allocs, 14,145,785 frees, 221,889,196 bytes allocated
==1350==
==1350== All heap blocks were freed -- no leaks are possible
==1350==
==1350== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

檢查doubly-linked list的function記憶體洩漏檢測，運行以下指令：
make
gcc -o memoryleak memoryleak.c -L. -ldatabase -Wl,-rpath=.
valgrind --leak-check=full --show-reachable=yes -s ./memoryleak

運行結果亦同，沒有任何記憶體洩漏。