

開放原始碼的回收與再利用

Qing

qing@cs.nthu.edu.tw

(Email/MSN)

2006/12/17

About Qing (1/2)

□ **Education**

- Ph.D. Candidate, Department of Computer Science, National Tsing-Hua University, Taiwan
- Research interests: distribute network management, mobile agent, VoIP, and p2p networking

□ **Software Development Skills**

- Programming languages: 80x86 assembly, C/C++, Java, C#
- J2EE development and Web programming: EJB, JSP/Servlet
- Network programming: TCP/IP, socket programming
- Object Oriented Design/Programming
- Design Patterns and Software Architecture
- Distributed Network Management System
- Peer-to-Peer Networking

About Qing (2/2)

□ **Honor**

- The champion of the Trend Micro Programming Contest 2004

□ **Recent Talks**

- JavaTwo Conference, 2003-2006

□ **Book Translation**

- Thinking in Java 2nd Edition, in Traditional Chinese
- Essential C++, in Traditional Chinese

Qing 現在在做什麼？

- 兩個 digg-like 的 service
- Musica
 - 一個音樂自動切割的軟體
- IPCam with PocketPC/Smartphone
- QRCode on PocketPC/Smartphone
- AirPreseneter
 - 無線投影
- 企業知識管理系統

Google 時代的程式撰寫

- Google 時代的來臨也加快了程式員的各種面向速度的提昇
 - 學習速度
 - 錯誤排除的速度
 - Google 是找到問題答案的最佳途徑，你會遇到的問題，別人多半也會
 - IM 軟體的流行也使得同儕網絡變成好的解題方式
 - 開發的速度
 - 開放原始碼質與量俱皆大增
 - 透過搜尋引擎極易取得所需的程式碼

前後 Google 時代的程式員的差異 (1/3)

- 學習的方式不同
 - 前 Google 時代的程式員：透過書籍或文章
 - 後 Google 時代的程式員：除了書籍或文章外，更從開放原始碼中學習
- 解決問題或麻煩的方式不同
 - 前 Google 時代的程式員：嘗試、摸索，詢問前輩，在線上社群或論壇 BBS 發問
 - 後 Google 時代的程式員：除了前 Google 時代的方法之外，更重視透過搜尋引擎尋找問題的原因、解法，甚至現成的程式碼

前後 Google 時代的程式員的差異 (2/3)

- 程式碼的來源不同
 - 前 Google 時代的程式員：一手打造
 - 後 Google 時代的程式員：除了自行撰寫必要的程式碼外，更善用網路上隨手可得的程式碼，加以裁切、添加、整合
- 重視的技能取向不同
 - 前 Google 時代的程式員：撰寫品質佳、易於重覆運用的程式碼

後 Google 時代的程式員需要的新技能

- 建立好的整合架構
- 善於搜尋所面臨問題的解決方案，並從中快速吸取新知
- 善於搜尋既有的程式碼
- 善於追蹤了解文件不足的程式碼
- 善於拆解、修繕既有的程式碼，以符合自己的目的

人們都拿開放原始碼專案做什麼？

- 使用開放原始碼專案產出的軟體
 - 大多數人
- 從開放原始碼專案中學習
- reuse 開放原始碼專案中的程式碼
- recycle 開放原始碼專案中的程式碼

Musica : 一個大量運用回收開放原始碼的專案

- Musica : 一個從公開音訊（例如數位廣播或網路廣播）做自動截斷歌曲的軟體
 - Winamp 的 iPod sync 模組
 - ffmpeg
 - MPlayer
 - LAME

Reuse vs. Recycle

□ Code Reuse

- 不需要碰觸到原始碼就可以達到運用的目的
- 需要原始程式碼設計完善夠彈性夠周延，或程式運氣夠好，遇到剛好滿足目的的程式碼

□ Code Recycle

- 程式碼來自世界各地，其目的或範圍往往不盡人意
- 需要施以回收再加工的作業，才能夠進一步加以利用

程式碼的重新打造 vs. 回收再生

- 對一名工匠而言，倘若要製作一張椅子，他可以
 - 找到原始的木材，重頭開始製作椅子
- 倘若有二手回收的家具，他也可以
 - 找到一張適用的木頭茶几
 - 重新修整這張茶几
 - 添加新的材料
 - 成為一張具有新面貌的椅子
- 學問在那？
 - 如何找到好的資源
 - 施以最小幅度的心力，達成相同的目的

程式碼回收與再利用的方法

- ❑ 在網路上搜尋可用的原始碼，並選擇最適合的
- ❑ 了解程式架構，拆解出自己所需的部份
- ❑ 訂定不同的階段，從最小的里程碑開始
- ❑ 做細部拆解，先求能編譯，再求正確執行
- ❑ 暫時忽略非第一個里程碑的內容
- ❑ 適度的斬斷關聯性，尋找適當的替代品
- ❑ 整理介面，去除不必要的元素
- ❑ 做好心理建設，面對混亂
- ❑ 跨出成功的第一步

在網路上搜尋可用的原始碼，並選擇最適合的

- 選擇執行平台相符或相似者
 - 相似時仍需花費移植心力
- 選擇程式語言相符者
 - Java 和 C# 之間互換較為容易
- 選擇版權宣告合適者
- 選擇範圍接近者
- 選擇使用者眾者
- 選擇相依性低的
 - 盡量不要選擇依賴程式庫多的專案

了解程式架構，拆解出自己所需的部份

- 即使已經盡量的選擇範圍接近自己所需的專案，但多出的部份仍需適當的拆解
- 拆解前要先了解程式架構
 - 閱讀能取得的文件
 - 程式碼的追蹤
- 如果不了解程式架構
 - 不知在此架構下的拆解的方式
 - 不知那些可以拆那些不能拆
 - 不知那些可以先不拆等日後再拆

訂定不同的階段，從最小的里程碑開始

- ❑ 在原始碼層次的拆解告一個段落後，程式碼通常連編譯都無法編譯
- ❑ 必須制定幾個不同階段的目標，並且從最小的里程碑開始出發
- ❑ 達到第一個里程碑具有十分重要的象徵性意義
 - 代表著你能夠正確的編譯而且執行（即使功能不太正確或功能尚不完備）
 - 但只要能夠達到第一個里程碑，之後的問題幾乎都能輕易迎刃而解

做細部拆解，先求能編譯，再求正確執行

- 在開放原始碼的回收與再利用活動中，最難的就是就是讓拆解下來的原始碼成功的編譯
- 拆解出來的部份往往相依於未被拆解的原專案組成
 - **header file** 中的定義或巨集
 - 原專案中的其他函式
- 為了成功的編譯，你需要再回到基礎專案中，把所需的部份拆解出來
 - 此即所謂細部拆解
- 細部拆解並不會直接把整個原始碼檔案移至回收專案中，僅移動在基礎專案中所需的部份
- 在這個階段，會反覆的看到許許多多的編譯錯誤
 - 逐一解決各個編譯上的錯誤
 - 需要的耐心耐心加耐心

暫時忽略非第一個里程碑的內容

- 矇上眼睛假裝看不見
 - 有許多編譯錯誤是第一個里程碑後才會需要的
 - 適時的將它們註解掉，以求通過編譯檢查是一個很好的手段
- 被註解掉的部份，在通過第一個里程碑後，解決其編譯錯誤的方案，也會在處理第一個里程碑的過程中產生
 - 此時再取消對它們的註解，就可以套用這些解決方案

適度的斬斷關聯性，尋找適當的替代品

- 許多開放原始碼的專案是盤根錯節
 - 有時編譯的錯誤是來自於缺少某個函式或類別，而如果要加入這個函式，必須引入一大堆東西
 - 必須考慮適度的斬斷和該函式的關聯性
 - Ex, `GList.c/GList.h` 相依於 `GLib` 其他部份
- 尋找替代品的兩個途徑
 - 自己撰寫
 - 再找現成的開放原始碼
- 運用替代品的方式
 - 修整介面，保持作用，符合目的

整理介面，去除不必要的元素

- 被回收的程式碼本身的目標往往和你運用的目標不盡相同
 - 其函式或類別的介面長相就會和你所需的不同
 - 介面中的元素會較我們所需的為多
- 因為拆解的關係，有許多編譯錯誤會來自於未含入介面中額外多出的元素
- 去除一些不必要的元素，並且重新整理這些介面的長相
- 得到第一個可以編譯的版本

做好心理建設，面對混亂

- 就算張開眼睛也仍然看不見
- 別人的程式碼習慣風格和你必然不同
- 不要浪費時間在重新整理上
- 請先做好心理建設，讓自己能夠面對自己覺得混亂的程式碼，視眼前之混亂如無物
- 有空閒時間，再套用 **refactoring** 的技巧，逐步的改善回收程式的結構

跨出成功的第一步

- 成功的達成第一個里程碑後，便逐一的依照同樣的方式加入
- 後續的里程碑達成的速度會愈來愈快

Case Study : 視訊檔案的格式探測

- 需求： 檢查給定的視訊檔案是否屬於特定的數種格式
- 限制： C/C++， 在 Win32 上執行， 編譯後大小必須在 100KB 以內

在網路上搜尋可用的原始碼，並選擇最適合的

- ❑ ffmpeg 有許多人使用，而且程式語言是 C，在 Win32 平台上編譯不致於有太多問題
- ❑ ffmpeg 的 libavformat 有提供此類的功能

了解程式架構，拆解出自己所需的部份

- 採用倒推法來了解程式架構
 - 所謂的倒推法，也就是從應用端倒推
- 先找到一段應用 libavformat 的範例

```
av_register_all();  
// Open video file  
if(av_open_input_file(&pFormatCtx, (char *) fileName, NULL, 0,  
    NULL)!=0 )  
    return false;  
// Retrieve stream information  
if(av_find_stream_info(pFormatCtx)<0)  
    return false; // Couldn't find stream information  
// Dump information about file onto standard error  
dump_format(pFormatCtx, 0, (char *) fileName, false);
```

倒推追蹤原始碼

- 善用 grep 工具（Win32 上可用 Windows Grep）
- 逐一檢查 dump_format() 前的各個函式

```
void av_register_all(void)
{
    static int inited = 0;
    if (inited != 0)
        return;
    inited = 1;
    avcodec_init();
    avcodec_register_all();
    mpegps_init();
    mpegts_init();
    ...
    /* 一堆 xxxx_init() */
}
```

倒推法就是 DFS — 深度優先搜尋

- 接下來往 mpegps_init() 展開，利用 grep 找出它在 mpeg.c 中

```
int mpegps_init(void)
{
#ifdef CONFIG_MUXERS
    av_register_output_format(&mpeg1system_mux);
    av_register_output_format(&mpeg1vcd_mux);
    av_register_output_format(&mpeg2vob_mux);
    av_register_output_format(&mpeg2svcd_mux);
    av_register_output_format(&mpeg2dvd_mux);
#endif //CONFIG_MUXERS
    av_register_input_format(&mpegps_demux);
    return 0;
}
```

續追 av_register_input_format()

```
void av_register_input_format(AVInputFormat *format)
{
    AVInputFormat **p;
    p = &first_ifformat;
    while (*p != NULL) p = &(*p)->next;
    *p = format;
    format->next = NULL;
}
```

看看 AVInputFormat 定義於何處

- 利用 grep 找出在 avformat.h

```
typedef struct AVInputFormat {
    const char *name;
    const char *long_name;
    int priv_data_size;
    int (*read_probe)(AVProbeData *);
    int (*read_header)(struct AVFormatContext *,
                      AVFormatParameters *ap);
    int (*read_packet)(struct AVFormatContext *, AVPacket *pkt);
    int (*read_close)(struct AVFormatContext *);
    int (*read_seek)(struct AVFormatContext *,
                    int stream_index, int64_t timestamp, int flags);
    int64_t (*read_timestamp)(struct AVFormatContext *s, int stream_index,
                              int64_t *pos, int64_t pos_limit);
    int flags;
    const char *extensions;
    int value;

    int (*read_play)(struct AVFormatContext *);
    int (*read_pause)(struct AVFormatContext *);
    struct AVInputFormat *next;
} AVInputFormat;
```

判斷 AVInputFormat 各欄位的作用

- 用非物件導向的方式來實作物件導向的多型
- 有很多一看就知道是探測格式時不會需要的
- 我們應該要特別注意 `read_probe` 及 `extension` 兩欄位

訂定不同的階段，從最小的里程碑開始

- 我們試著訂出三個階段
 - 探測 mpegps 格式
 - 探測 mpegts 格式
 - 探測其他格式
- 試著編譯，一定會發生很多問題
 - 結構之間的環環相扣：為了 AVInputFormat 中的 read_header，我們得括入 AVFormatContext 和 AVFormatParameters
- 後來我們會因為更了解而發現這個函式指標根本不需要
 - 但此刻的我們只好先把它們先加入
- 在 utils.c 中的衆多函式同樣的也被清的只剩下：
av_register_input_format()、match_ext()、av_probe_input_format()

適度的斬斷關聯性，尋找適當的替代品

- `av_probe_input_format()` 會用到許多 utility 函式
 - `av_realloc()`、`get_buffer()`、`url_open()`、`url_fseek()`、`url_fclose()`
- 這些函式看起來就是標準 C 函式庫中的函式差不多。所以決定退化回標準 C 函式庫中函式
 - 改用 `malloc()` 來代替 `av_realloc()` 及 `get_buffer()` 的作用；改用 `fopen()` 來代替 `url_open()`；改用 `fseek()` 來代替 `url_fseek()`；改用 `fclose()` 來代指 `url_fclose()`
- 要建立防火牆，不能讓相依性的火一直蔓延的燒

整理介面，去除不必要的元素


- 在這過程中，我們愈來愈了解整體的運作方式
- 所以我們發現 AVInputFormat 裡的結構有很多欄位是不必要的
 - 只需要保用 read_probe 這個函式指標即可
 - 其他的部份我們都可以拿掉
 - 更可以順手拿到一些不必要的資料結構定義
 - 例如拿掉了 read_header 可以一舉拿掉 AVFormatContext 及 AVFormatParameters 的定義

做好心理建設，面對混亂

- 在這個過程中，我既不去修改它的命名慣例，更不會去重新排版
- 我把所有取出來的程式碼包在一塊，把它當做一個黑箱子來使用
- 有些取自標頭檔的定義也許還有精簡的可能性，但是先將它擱在一塊
- 此刻我們需要的只是生產力—短期間內透過回收和整理取得我們想要的產物

結論

- ❑ 最後支援十種格式以上的這個格式探測程式庫，大小不到 40KB
- ❑ 幾乎沒有寫下半行程式碼，但憑藉著回收和整理的技巧，只需一晚，我們得到了一個相當實用的程式庫
- ❑ 原先對各視訊檔案格式完全不通的我，也能達成這個目標，而且在這個過程中對視訊檔案格式也有一定的認識
- ❑ 開放原始碼的各種程式片段俯拾可得
- ❑ 需要的是化桌子為椅子的功夫



Q & A

Thanks