智慧指標

Mudream

忘了delete....

這會把垃圾留在記憶體裡面

留垃圾在記憶體裡面

```
void foo(int par1, ...){
    //bla...
    //bla...
    Something* sth = new Something;
    //bla...
    return;
    // forget to delete sth!!!
}
```

- Call一次可能還不會出事，但多Call幾次，記憶體沒釋放好那就會GG了！

啊不就補個`delete`就沒事了？

- 可是假如在函數裡`new`很多東西，那不小心少掉一個的機率會有點高。

```
Foo1* foo1 = new Foo1;
Foo2* foo2 = new Foo2;
Foo3* foo3 = new Foo3;
Foo4* foo4 = new Foo4;
Foo5* foo5 = new Foo5;
Foo6* foo6 = new Foo6;
```

- STL裡有個東西可以把這發生的機率降到零
- 因為用了他後，不需要特別寫`delete`

STL: unique_ptr

引用計數

計算一個指標被引用的次數

情境

- 假想遊戲裡面要維護很多關卡，每個關卡用到的圖片不盡相同

```
struct Bitmap{
    //一些資料...
};

struct Stage{
    Bitmap* wall_tile;
    // 每個關卡可能會用到不一樣的牆壁圖片
    //...
};

Stage* stage[100];
```

假如只有`stage[1]`、`stage[2]`、`stage[4]`引用`frozen_wall`

```
stage[1] => frozen_wall
stage[2] => frozen_wall
stage[4] => frozen_wall
```

- 然後又希望可以在離開每個關卡時**適當**的釋放資源
- 因為關卡很多，所以希望在玩家離開該關卡時，就先把資源釋放掉。

釋放？不是只要`delete stage[i]`嗎？

- 可是Bitmap是new出來的指標，太多可能會讓記憶體爆炸。

刪除的時機

- 假設玩家進關順序是這樣的

```
stage1 => stage2 => stage4 => stage5
```

- 刪除stage1時，該把frozen_wall `delete`嗎？

```
不該，因為還有`stage2` `stage4`
```

- 刪除stage2時，該把frozen_wall `delete`嗎？

```
不該，因為還有`stage4`
```

- 刪除stage4時，該把frozen_wall `delete`嗎？

```
**可以**，沒有其他人引用他了。
```

天啊 這有點崩潰

Solution: 算引用次數

- 在Bitmap裡多維護一個`int`，計算被引用的次數
- 在每次stage要引用時，就幫他`+1`
- 在每次stage要被刪除時，就幫他`-1`

  並且檢查是否為零，假如是零，那就移除

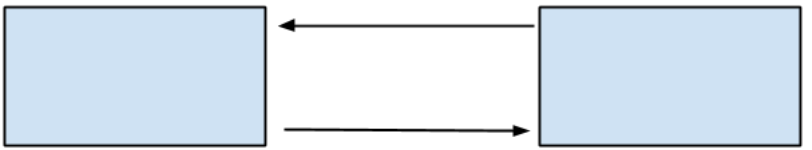
STL: shared_ptr

當我們圍成一圈

share_ptr 在這情況還是無能為力

引用計數還是有機會爆炸

概念：循環引用



```
struct A{B* b};
struct B{A* a};
A* aa = new aa;
B* bb = new bb;
aa->b = bb, b->a = aa;
```

- 在沒有其他指標指向他們，他們的引用計數還是非零，因為他們互相指。

解決方案

讓其中一個引用不會真的增加引用計數。

STL: weak_ptr

回顧

記憶體管理

把忘了delete發生機率降為零

引用計數

環狀引用解決方案