設計模式C++實現（９）——享元模式

星期六, 2013 12月 14, 1:06 上午

軟件領域中的設計模式為開發人員提供了一種使用專家設計經驗的有效途徑。設計模式中運用了面向對象編程語言的重要特性：封裝、繼承、多態，真正領悟設計模式的精髓是可能一個漫長的過程，需要大量實踐經驗的積累。最近看設計模式的書，對於每個模式，用C++寫了個小例子，加深一下理解。主要參考《大話設計模式》和《設計模式:可復用面向對象軟件的基礎》（DP）兩本書。本文介紹享元模式的實現。

舉個圍棋的例子，圍棋的棋盤共有361格，即可放361個棋子。現在要實現一個圍棋程序，該怎麼辦呢？首先要考慮的是棋子棋盤的實現，可以定義一個棋子的類，成員變量包括棋子的顏色、形狀、位置等信息，另外再定義一個棋盤的類，成員變量中有個容器，用於存放棋子的對象。下面給出代碼表示：

棋子的定義，當然棋子的屬性除了顏色和位置，還有其他的，這裡略去。這兩個屬性足以說明問題。

```
1.  //棋子顏色
2.  enum PieceColor {BLACK, WHITE};
3.  //棋子位置
4.  struct PiecePos
5.  {
6.      int x;
7.      int y;
8.      PiecePos(int a, int b): x(a), y(b) {}
9.  };
10. //棋子定義
11. class Piece
12. {
13. protected:
14.     PieceColor m_color; //顏色
15.     PiecePos m_pos;     //位置
16. public:
17.     Piece(PieceColor color, PiecePos pos): m_color(color), m_pos(pos) {}
18.     ~Piece() {}
19.     virtual void Draw() {}
20. };
21. class BlackPiece: public Piece
22. {
23. public:
24.     BlackPiece(PieceColor color, PiecePos pos): Piece(color, pos) {}
25.     ~BlackPiece() {}
26.     void Draw() { cout<<"繪製一顆黑棋"<<endl;}
27. };
28. class WhitePiece: public Piece
29. {
30. public:
31.     WhitePiece(PieceColor color, PiecePos pos): Piece(color, pos) {}
32.     ~WhitePiece() {}
33.     void Draw() { cout<<"繪製一顆白棋"<<endl;}
34. };
```

棋盤的定義：

```
1.  class PieceBoard
2.  {
3.  private:
4.      vector<Piece*> m_vecPiece; //棋盤上已有的棋子
5.      string m_blackName; //黑方名稱
6.      string m_whiteName; //白方名稱
7.  public:
8.      PieceBoard(string black, string white): m_blackName(black), m_whiteName(white)
    {}
9.      ~PieceBoard() { Clear(); }
10.     void SetPiece(PieceColor color, PiecePos pos) //一步棋，在棋盤上放一顆棋子
11.     {
12.         Piece * piece = NULL;
13.         if(color == BLACK) //黑方下的
14.         {
15.             piece = new BlackPiece(color, pos); //獲取一顆黑棋
16.             cout<<m_blackName<<"在位置("<<pos.x<<′,′<<pos.y<<")";
17.             piece->Draw(); //在棋盤上繪製出棋子
18.         }
19.         else
20.         {
21.             piece = new WhitePiece(color, pos);
22.             cout<<m_whiteName<<"在位置("<<pos.x<<′,′<<pos.y<<")";
23.             piece->Draw();
24.         }
25.         m_vecPiece.push_back(piece);  //加入容器中
26.     }
27.     void Clear() //釋放內存
28.     {
29.         int size = m_vecPiece.size();
30.         for(int i = 0; i < size; i++)
31.             delete m_vecPiece[i];
32.     }
33. };
```

客戶的使用方式如下：

```
1.  int main()
2.  {
3.      PieceBoard pieceBoard("A","B");
4.      pieceBoard.SetPiece(BLACK, PiecePos(4, 4));
5.      pieceBoard.SetPiece(WHITE, PiecePos(4, 16));
6.      pieceBoard.SetPiece(BLACK, PiecePos(16, 4));
7.      pieceBoard.SetPiece(WHITE, PiecePos(16, 16));
8.  }
```

可以發現，棋盤的容器中存放了已下的棋子，而每個棋子包含棋子的所有屬性。一盤棋往往需要含上百顆棋子，採用上面這種實現，佔用的空間太大了。如何改進呢？用享元模式。其定義為：運用共享技術有效地支持大量細粒度的對象。

在圍棋中，棋子就是大量細粒度的對象。其屬性有內在的，比如顏色、形狀等，也有外在的，比如在棋盤上的位置。內在的屬性是可以共享的，區分在於外在屬性。因此，可以這樣設計，只需定義兩個棋子的對象，一顆黑棋和一顆白棋，這兩個對象含棋子的內在屬性；棋子的外在屬性，即在棋盤上的位置可以提取出來，存放在單獨的容器中。相比之前的方案，現在容器中僅僅存放了位置屬性，而原來則是棋子對象。顯然，現在的方案大大減少了對於空間的需求。

關注PieceBoard 的容器，之前是vector<Piece*> m_vecPiece，現在是vector<PiecePos> m_vecPos。這裡是關鍵。

棋子的新定義，只包含內在屬性：

```
1. //棋子顏色
2. enum PieceColor {BLACK, WHITE};
3. //棋子位置
4. struct PiecePos
5. {
6.     int x;
7.     int y;
8.     PiecePos(int a, int b): x(a), y(b) {}
9. };
10. //棋子定義
11. class Piece
12. {
13. protected:
14.     PieceColor m_color; //顏色
15. public:
16.     Piece(PieceColor color): m_color(color) {}
17.     ~Piece() {}
18.     virtual void Draw() {}
19. };
20. class BlackPiece: public Piece
21. {
22. public:
23.     BlackPiece(PieceColor color): Piece(color) {}
24.     ~BlackPiece() {}
25.     void Draw() { cout<<"繪製一顆黑棋\n"; }
26. };
27. class WhitePiece: public Piece
28. {
29. public:
30.     WhitePiece(PieceColor color): Piece(color) {}
31.     ~WhitePiece() {}
32.     void Draw() { cout<<"繪製一顆白棋\n";}
33. };
```

相應棋盤的定義為：

```
1. class PieceBoard
2. {
3. private:
4.     vector<PiecePos> m_vecPos; //存放棋子的位置
5.     Piece *m_blackPiece;       //黑棋棋子
6.     Piece *m_whitePiece;       //白棋棋子
7.     string m_blackName;
8.     string m_whiteName;
9. public:
10.     PieceBoard(string black, string white): m_blackName(black), m_whiteName(white)
11.     {
12.         m_blackPiece = NULL;
13.         m_whitePiece = NULL;
14.     }
15.     ~PieceBoard() { delete m_blackPiece; delete m_whitePiece;}
```

```
16.     void SetPiece(PieceColor color, PiecePos pos)
17.     {
18.         if(color == BLACK)
19.         {
20.             if(m_blackPiece == NULL) //只有一顆黑棋
21.                 m_blackPiece = new BlackPiece(color);
22.             cout<<m_blackName<<"在位置("<<pos.x<<','<<pos.y<<")";
23.             m_blackPiece->Draw();
24.         }
25.         else
26.         {
27.             if(m_whitePiece == NULL)
28.                 m_whitePiece = new WhitePiece(color);
29.             cout<<m_whiteName<<"在位置("<<pos.x<<','<<pos.y<<")";
30.             m_whitePiece->Draw();
31.         }
32.         m_vecPos.push_back(pos);
33.     }
34. };
```

客戶的使用方式一樣，這裡不重複給出，現在給出享元模式的UML圖，以圍棋為例。棋盤中含兩個共享的對象，黑棋子和白棋子，所有棋子的外在屬性都存放在單獨的容器中。