# 1、背景

在软件开发中存在着这样复杂的对象，它们拥有一系列成员属性，这些成员属性中有些是引用类型的成员对象。并且，在这些复杂对象中，还可能存在一些限制条件，如某些属性没有赋值则复杂对象不能作为一个完整的产品使用；有些属性的赋值必须按照一定的顺序，一个属性没有赋值之前，另一个属性可能无法赋值等。

复杂对象相当于一辆待建造的汽车，而对象的属性相当于汽车的部件，建造产品的过程就相当于组合部件的过程。由于组合部件的过程很复杂，因此这些部件的组合过程往往被“外部化”到一个称作建造者的对象里，建造者返回给客户端的是一个已经构建完毕的完整的产品对象，而用户无需关心该对象所包含的属性以及它们的组装方式，这就是建造者模式存在的背景。

# 2、定义

建造者模式（Builder Pattern）：将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示。

建造者模式是一步一步创建一个复杂的对象，它允许用户只通过指定复杂对象的类型和内容就可以构建它们，用户不需要知道内部的具体构建细节。建造者模式属于对象创建型模式，又称为生成器模式。

# 3、结构

建造者模式包含以下角色：

Builder：抽象建造者

ConcreteBuilder：具体建造者

Director：指挥者

Product：产品角色

# 4、代码

#include <iostream>

#include "ConcreteBuilder.h"

#include "Director.h"

#include "Builder.h"

#include "Product.h"

using namespace std;

int main(int argc, char \*argv[])

{

ConcreteBuilder \* builder = new ConcreteBuilder();

Director director;

director.setBuilder(builder);

Product \* pd = director.constuct();

pd->show();

delete builder;

delete pd;

return 0;

}

///////////////////////////////////////////////////////////

// ConcreteBuilder.cpp

// Implementation of the Class ConcreteBuilder

// Created on: 02-十月-2014 15:57:03

// Original author: colin

///////////////////////////////////////////////////////////

#include "ConcreteBuilder.h"

ConcreteBuilder::ConcreteBuilder(){

}

ConcreteBuilder::~ConcreteBuilder(){

}

void ConcreteBuilder::buildPartA(){

m\_prod->setA("A Style "); //不同的建造者，可以实现不同产品的建造

}

void ConcreteBuilder::buildPartB(){

m\_prod->setB("B Style ");

}

void ConcreteBuilder::buildPartC(){

m\_prod->setC("C style ");

}

///////////////////////////////////////////////////////////

// Director.cpp

// Implementation of the Class Director

// Created on: 02-十月-2014 15:57:01

// Original author: colin

///////////////////////////////////////////////////////////

#include "Director.h"

Director::Director(){

}

Director::~Director(){

}

Product\* Director::constuct(){

m\_pbuilder->buildPartA();

m\_pbuilder->buildPartB();

m\_pbuilder->buildPartC();

return m\_pbuilder->getResult();

}

void Director::setBuilder(Builder\* buider){

m\_pbuilder = buider;

}

# 5、分析

# 6、实例

# 7、优缺点

# 8、应用