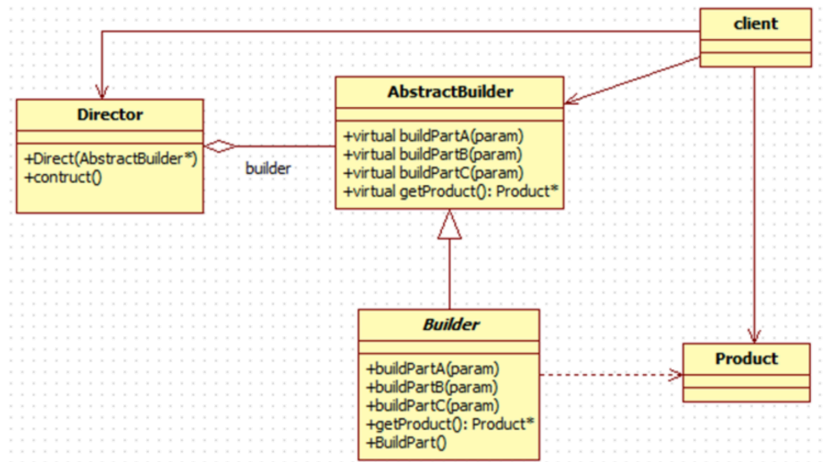


**动机 (Motivation)** :在软件系统中，有时候面临着“一个复杂对象”的创建工作，其通常由各个部分的子对象用一定的算法构成；由于需求的变化，这个复杂对象的各个部分经常面临着剧烈的变化，但是将它们组合在一起的算法却相对稳定。如何应对这种变化？如何提供一种“封装机制”来隔离出“复杂对象的各个部分”的变化，从而保持系统中的“稳定构建算法”不随着需求改变而改变？

定义:将一个复杂对象的构建与其表示相分离，使得同样的构建过程(稳定)可以创建不同的表示(变化)。——《设计模式》GoF



要点总结:Builder 模式主要用于“分步骤构建一个复杂的对象”。在这其中“分步骤”是一个稳定的算法，而复杂对象的各个部分则经常变化。变化点在哪里，封装哪里—— Builder模式主要在于应对“复杂对象各个部分”的频繁需求变动。其缺点在于难以应对“分步骤构建算法”的需求变动。

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4  class Vehicle{
5  public:
6      Vehicle(){
7          strBody = "";
8          strTyre = "";
9          strEngine = "";
10     }
11     void Show(){
12         if(strBody == "" || strTyre == "" || strEngine == ""){
13             cout << "车辆还未建造" << endl;
14             return;
15         }
16         cout << "车辆建造完成: " << strBody << ", " << strTyre << ", " << strEngine << endl;
17     }
18 public:
19     string strBody;//车身,轮胎,引擎
20     string strTyre;
21     string strEngine;
22 };
23 class BuilderVehicle{
24 public:
25     virtual ~BuilderVehicle(){}
26     virtual void BuildBody() = 0;
27     virtual void BuildTyre() = 0;
28     virtual void BuildEngine() = 0;
29 };
30 class BuilderSUV : public BuilderVehicle{
31 private:

```

```

32     Vehicle * m_pVehicle;
33 public:
34     BuilderSUV(Vehicle *pV) : m_pVehicle(pV){}
35     void BuildBody(){
36         m_pVehicle->strBody = "SUV 车身";
37     }
38     void BuildTyre(){
39         m_pVehicle->strTyre = "SUV 轮胎";
40     }
41     void BuildEngine(){
42         m_pVehicle->strEngine = "SUV 引擎";
43     }
44 };
45 class BuilderTruck : public BuilderVehicle{
46 private:
47     Vehicle * m_pVehicle;
48 public:
49     BuilderTruck(Vehicle *pV) : m_pVehicle(pV){}
50     void BuildBody(){
51         m_pVehicle->strBody = "卡车 车身";
52     }
53     void BuildTyre(){
54         m_pVehicle->strTyre = "卡车 轮胎";
55     }
56     void BuildEngine(){
57         m_pVehicle->strEngine = "卡车 引擎";
58     }
59 };
60 class Director{
61 public:
62     Director(){}
63     void Build(BuilderVehicle *pBuild){
64         pBuild->BuildBody();
65         pBuild->BuildTyre();
66         pBuild->BuildEngine();
67     }
68 };
69 int main(){
70     Vehicle V;
71     BuilderTruck Truck(&V);
72     Director D;
73     D.Build(&Truck);
74     V.Show();
75
76     BuilderSUV SUV(&V);
77     D.Build(&SUV);
78     V.Show();
79     return 0;
80 }

```

```

1 #include <iostream>
2 #include <vector>
3 #include <string>

```

```

4 using namespace std;
5 class Product{
6 private:
7     vector<string> vecParts;
8 public:
9     void AddAPart(const string strPart){
10         vecParts.push_back(strPart);
11     }
12     virtual void Show() {
13         for(int i = 0; i<vecParts.size(); i++){
14             cout << vecParts[i] << " , ";
15         }
16         vecParts.clear();
17         cout << endl;
18     }
19 };
20 class Builder{
21 public:
22     virtual void BuildHead() = 0;
23     virtual void BuildBody() = 0;
24     virtual void BuildHand() = 0;
25     virtual void BuildFeet() = 0;
26     virtual ~Builder(){}
27     virtual Product* GetProduct() = 0;
28 };
29 class FatPersonBuilder : public Builder{
30 private:
31     Product * m_pProduct;
32 public:
33     FatPersonBuilder(Product* pPro):m_pProduct(pPro){}
34     void BuildHead() {
35         m_pProduct->AddAPart("胖子的头");
36     }
37     void BuildBody() {
38         m_pProduct->AddAPart("胖子的身体");
39     }
40     void BuildHand() {
41         m_pProduct->AddAPart("胖子的手");
42     }
43     void BuildFeet() {
44         m_pProduct->AddAPart("胖子的脚");
45     }
46     Product* GetProduct(){
47         return m_pProduct;
48     }
49 };
50 class ThinPersonBuilder : public Builder{
51 private:
52     Product * m_pProduct;
53 public:
54     ThinPersonBuilder(Product* pPro):m_pProduct(pPro){}
55     void BuildHead() {
56         m_pProduct->AddAPart("瘦子的头");
57     }

```

```

58     void BuildBody() {
59         m_pProduct->AddAPart("瘦子的身体");
60     }
61     void BuildHand() {
62         m_pProduct->AddAPart("瘦子的手");
63     }
64     void BuildFeet() {
65         m_pProduct->AddAPart("瘦子的脚");
66     }
67     Product* GetProduct(){
68         return m_pProduct;
69     }
70 };
71 class Director{
72 public:
73     Director(){}
74     void CreatPerson(Builder *pBuilder)
75     {
76         pBuilder->BuildHead();
77         pBuilder->BuildBody();
78         pBuilder->BuildHand();
79         pBuilder->BuildFeet();
80     }
81 };
82 int main(){
83     Product objProduct;
84     FatPersonBuilder *pFat = new FatPersonBuilder(&objProduct);
85     ThinPersonBuilder *pThin = new ThinPersonBuilder(&objProduct);
86     Director objDirector;
87     objDirector.CreatPerson(pFat);
88     pFat->GetProduct()->Show();
89     objDirector.CreatPerson(pThin);
90     pThin->GetProduct()->Show();
91     delete pFat;
92     delete pThin;
93     return 0;
94 }

```

```

192:DesignPattnsStudy weishichun$ ls Builder
Builder_1.cpp      Builder_2.cpp      Builder构建器.pdf
192:DesignPattnsStudy weishichun$ g++ -o Builder1.out Builder_1.cpp
192:DesignPattnsStudy weishichun$ ./Builder1.out
车辆建造完成：卡车 车身，卡车 轮胎，卡车 引擎
车辆建造完成：SUV 车身，SUV 轮胎，SUV 引擎
192:DesignPattnsStudy weishichun$ g++ -o Builder2.out Builder_2.cpp
192:DesignPattnsStudy weishichun$ ./Builder2.out
胖子的头，胖子的身体，胖子的手，胖子的脚，
瘦子的头，瘦子的身体，瘦子的手，瘦子的脚，
192:DesignPattnsStudy weishichun$

```