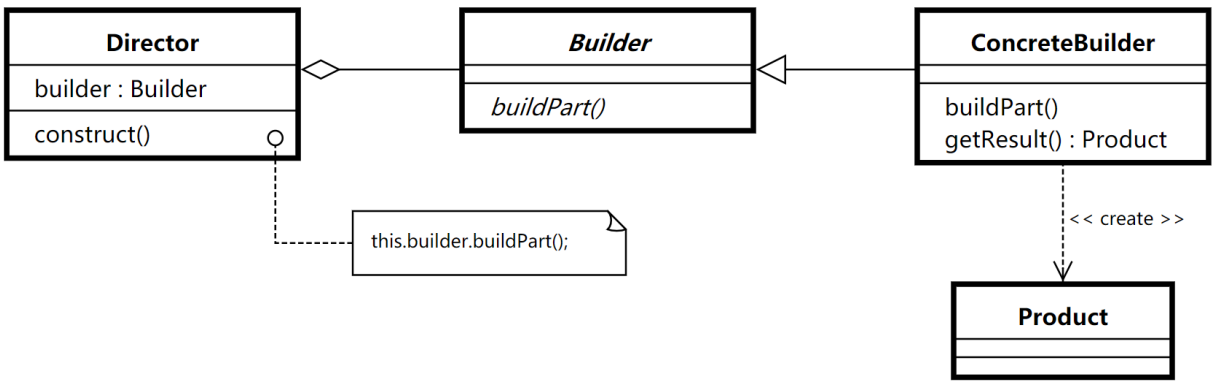
建造者模式

郭嘉

模式定义：

将一个复杂对象的创建与他的表示分离，使得同样的构建过程可以创建不同的表示



建造者模式实例

```
1 package com.tuling.designpattern.builder;
2
```

```java
/**
 * @author 腾讯课堂-图灵学院 郭嘉
 * @Slogan 致敬大师，致敬未来的你
 */
public class BuilderTest {
 public static void main(String[] args) {
// Product product=new Product( );
 // product.setCompanyName( "xxx" );
 // product.setPart1( "xxx" );
 // // ...
 //
 // // ....
 // // ....
 //

  ProductBuilder specialConcreteProductBuilder=new SpecialConcreteProductBuilder();
  Director director=new Director(specialConcreteProductBuilder);
  Product product=director.makeProduct( "productNamexxx", "cn...", "part1", "part2", "part3", "part4" );
  System.out.println(product);


  }
}

interface ProductBuilder{
 void builderProductName(String productName);
 void builderCompanyName(String companyName);
 void builderPart1(String part1);
 void builderPart2(String part2);
 void builderPart3(String part3);
 void builderPart4(String part4);

 Product build();
}

class DefaultConcreteProductBuilder implements ProductBuilder{
 private String productName;
 private String companyName;
 private String part1;
```

```
42   private String part2;
43   private String part3;
44   private String part4;
45   @Override
46   public void builderProductName(String productName) {
47   this.productName=productName;
48   }
49
50   @Override
51   public void builderCompanyName(String companyName) {
52   this.companyName=companyName;
53   }
54
55   @Override
56   public void builderPart1(String part1) {
57   this.part1=part1;
58   }
59
60   @Override
61   public void builderPart2(String part2) {
62   this.part2=part2;
63   }
64
65   @Override
66   public void builderPart3(String part3) {
67   this.part3=part3;
68   }
69
70   @Override
71   public void builderPart4(String part4) {
72   this.part4=part4;
73   }
74
75   @Override
76   public Product build() {
77   return new Product( this.productName,this.companyName,this.part1,this.pa
rt2,this.part3,this.part4 );
78   }
79 }
80
81 class SpecialConcreteProductBuilder implements ProductBuilder{
```

```
82  private String productName;
83  private String companyName;
84  private String part1;
85  private String part2;
86  private String part3;
87  private String part4;
88  @Override
89  public void builderProductName(String productName) {
90  this.productName=productName;
91  }
92
93  @Override
94  public void builderCompanyName(String companyName) {
95  this.companyName=companyName;
96  }
97
98  @Override
99  public void builderPart1(String part1) {
100  this.part1=part1;
101  }
102
103  @Override
104  public void builderPart2(String part2) {
105  this.part2=part2;
106  }
107
108  @Override
109  public void builderPart3(String part3) {
110  this.part3=part3;
111  }
112
113  @Override
114  public void builderPart4(String part4) {
115  this.part4=part4;
116  }
117
118  @Override
119  public Product build() {
120  return new Product( this.productName,this.companyName,this.part1,this.p
art2,this.part3,this.part4 );
121  }
```

```
122 }
123 class Director{
124
125  private ProductBuilder builder;
126
127  public Director(ProductBuilder builder) {
128  this.builder=builder;
129  }
130
131  public Product makeProduct(String productName, String companyName, Stri
ng part1, String part2, String part3, String part4){
132
133  builder.builderProductName(productName );
134  builder.builderCompanyName(companyName );
135  builder.builderPart1(part1 );
136
137  builder.builderPart2(part2 );
138  builder.builderPart3(part3 );
139  builder.builderPart4(part4 );
140
141  Product product=builder.build();
142  return product;
143
144  }
145
146
147 }
148
149 class Product {
150
151  private String productName;
152  private String companyName;
153  private String part1;
154  private String part2;
155  private String part3;
156  private String part4;
157  // ......
158
159
160  public Product() {
161  }
162
```

```
163   public Product(String productName, String companyName, String part1, St
ring part2, String part3, String part4) {
164   this.productName=productName;
165   this.companyName=companyName;
166   this.part1=part1;
167   this.part2=part2;
168   this.part3=part3;
169   this.part4=part4;
170   }
171
172   public String getProductName() {
173   return productName;
174   }
175
176   public void setProductName(String productName) {
177   this.productName=productName;
178   }
179
180   public String getCompanyName() {
181   return companyName;
182   }
183
184   public void setCompanyName(String companyName) {
185   this.companyName=companyName;
186   }
187
188   public String getPart1() {
189   return part1;
190   }
191
192   public void setPart1(String part1) {
193   this.part1=part1;
194   }
195
196   public String getPart2() {
197   return part2;
198   }
199
200   public void setPart2(String part2) {
201   this.part2=part2;
202   }
```

```
203
204   public String getPart3() {
205   return part3;
206   }
207
208   public void setPart3(String part3) {
209   this.part3=part3;
210   }
211
212   public String getPart4() {
213   return part4;
214   }
215
216   public void setPart4(String part4) {
217   this.part4=part4;
218   }
219
220   @Override
221   public String toString() {
222   return "Product{" +
223   "productName='" + productName + '\'' +
224   ", companyName='" + companyName + '\'' +
225   ", part1='" + part1 + '\'' +
226   ", part2='" + part2 + '\'' +
227   ", part3='" + part3 + '\'' +
228   ", part4='" + part4 + '\'' +
229   '}';
230   }
231 }
```

建造者模式与不可变对象配合使用

```
1  public class ProductTest2 {
2   public static void main(String[] args) {
3   Product.Builder builder=new Product.Builder().productName( "xxxx" ).compa
nyName( "xxxxxx" ).part1( "xxxxx" ).part2( "xxxx" );
4
5   //
6   builder.part3( "part3" );
7
```

```
 Product product=builder.build();
 System.out.println(product);
  }

}
class Product {

  private final String productName;
  private final String companyName;
  private final String part1;
  private final String part2;
  private final String part3;
  private final String part4;
  // ......



  public Product(String productName, String companyName, String part1, Str
ing part2, String part3, String part4) {
  this.productName=productName;
  this.companyName=companyName;
  this.part1=part1;
  this.part2=part2;
  this.part3=part3;
  this.part4=part4;
  }



  static class Builder{

  private String productName;
  private String companyName;
  private String part1;
  private String part2;
  private String part3;
  private String part4;


  public Builder productName(String productName){
  this.productName=productName;
```

```
49  return this;
50  }
51  public Builder companyName(String companyName){
52  this.companyName=companyName;
53  return this;
54  }
55  public Builder part1(String part1){
56  this.part1=part1;
57  return this;
58  }
59  public Builder part2(String part2){
60  this.part2=part2;
61  return this;
62  }
63  public Builder part3(String part3){
64  this.part3=part3;
65  return this;
66  }
67  public Builder part4(String part4){
68  this.part4=part4;
69  return this;
70  }
71
72  Product build(){
73  //
74  Product product=new Product( this.productName, this.companyName, this.pa
rt1, this.part2, this.part3, this.part4 );
75  return product;
76  }
77
78
79  }
80
81
82  @Override
83  public String toString() {
84  return "Product{" +
85  "productName='" + productName + '\'' +
86  ", companyName='" + companyName + '\'' +
87  ", part1='" + part1 + '\'' +
88  ", part2='" + part2 + '\'' +
```

```
  ", part3='" + part3 + '\'' +
  ", part4='" + part4 + '\'' +
  '}';
  }
}
```

应用场景

1. 需要生成的对象具有复杂的内部结构
2. 需要生成的对象内部属性本身相互依赖
3. 与不可变对象配合使用

优点:

1、建造者独立，易扩展。

2、便于控制细节风险。

Spring源码中的应用

```
org.springframework.web.servlet.mvc.method.RequestMappingInfo
org.springframework.beans.factory.support.BeanDefinitionBuilder
```