builder模式

场景

（1）构造一个复杂的对象，很多的属性，有些属性构造的时候需要做一些校验，格式转换

public class Product {

   private String field1;

   private String field2;

}

public **interface Builder** {

   Builder field1(String value);

Builder field2(String value);

Product create();

}

public class **ConcreteBuilder implements Builder** {

   private Product product = new Product();

   private ConcreteBuilder() {

  }

   public static Builder build() {

       return ConcreteBuilder();

  }

   public Product create() {

       return product;

  }

   public Builder field1(String value) {

       product.setField1(value);

       return this;

  }

   public Builder field2(String value) {

       product.setField2(value);

       return this;

  }

}

public class BuilderFactory {

   public static Builder build() {

       return new ConcreteBuilder();

  }

}

public class Client {

   public static void main(String[] args) {

       Builder builder = BuilderFactory.build();

       Product product = builder.field1("属性1").field2("属性2").create();

       System.out.println(product);

  }

}

构造器是一种非常棒，非常实用，非常常用的设计模式，大家务必要掌握好

常见于在构建一个复杂的对象，或者是构建一个复杂的表达式的时候，在开源框架中有大量广泛的运用

给大家举个例子，Mockito框架中，when().thenReturn()之类的，其实就是构造器模式的一种，通过多个连续的方法完成一个模拟对象的构建；还有就是spring test框架汇总，MvcMock对象，也是通过thenExpected()等多个连续的方法，完成一个mvc测试对象的构建，这块我们首先在后面spring boot源码剖析的时候，会带着大家来阅读这些源码，掌握实践。同时在阶段一的电商系统设计中，我们对于一些复杂对象的构建，给大家举个例子，比如说销售出库单的构建，如此复杂的对象，采用构造器模式是非常合适的

用构造器模式的最大好处是啥？复杂对象的构建过程太复杂了，里面可能包含一些业务逻辑，比如值检查，格式转换之类的。如果每个客户端都自己手动去完成构建的话，那么大量的冗余代码是一个；另外一个，如果我们要改变对象的构建过程的实现，可以就在构造器中一个地方修改即可，对于调用方完全透明；最后，如果我们要完全替换掉以前的builder实现的话，那么完全可以在工厂里替换一个实现，还跟简单工厂模式结合起来使用了

853769620