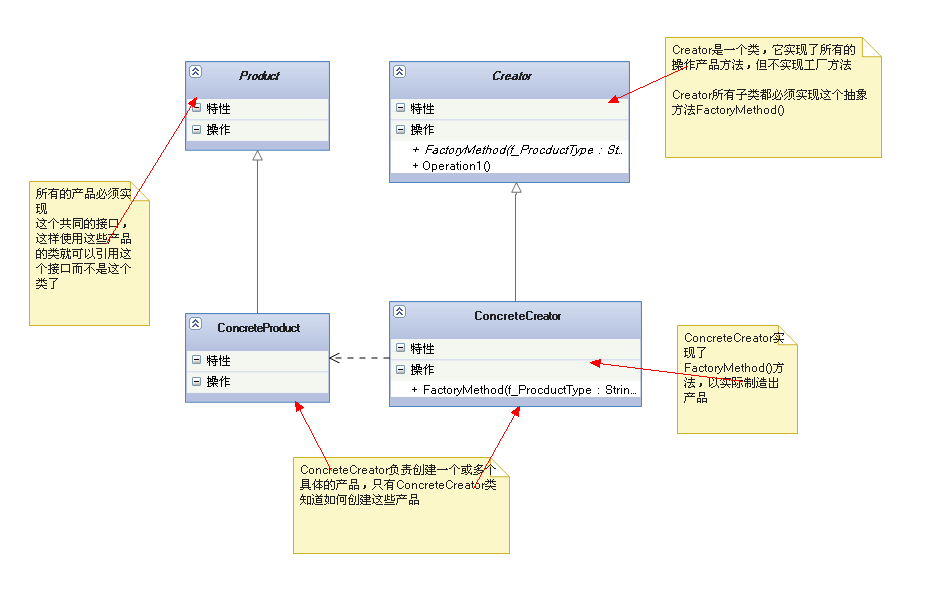
今天说一下工厂方法模式：

定义了一个创建对象的接口，但由子类决定要实例化的类是哪一个。工厂方法让类把实例化推迟到子类

所谓的决定并不是批模式允许子类本身在运行时做决定，而是指在编写创建者类时，不需知道创建的产品是哪一下，选择了使用

哪个子类，就决定了实际创建的产品是什么。



[复制代码](javascript:void(0);)

1 #region 工厂模式

2

3 // 产品

4 public abstract class Product

5 {

6 public string productName;

7 }

8

9 // 建造者

10 //工厂方法是创建一个框架，让子类决定要如何实现具体的产品

11 public abstract class Creator

12 {

13 public Product FactoryMethod(string f\_ProductType)

14 {

15 Product \_product;

16 \_product=CreateProduct(f\_ProductType);  
　　　　　　　　//可对产品做其它的操作......

17 return \_product;

18 }

19 //让子类去实现要生产什么产品

20 public abstract Product CreateProduct(string f\_Type);

21

22 }

23 #region 产品

24 public class OneProduct : Product

25 {

26 public OneProduct()

27 {

28 productName = "OneProduct";

29 }

30 }

31

32 public class TwoProduct : Product

33 {

34 public TwoProduct()

35 {

36 productName = "TwoProduct";

37 }

38 }

39

40 public class FirstProduct : Product

41 {

42 public FirstProduct()

43 {

44 productName = "My FirstProduct";

45 }

46 }

47

48 public class SecondProduct : Product

49 {

50 public SecondProduct()

51 {

52 productName = "My SecondProduct";

53 }

54 }

55 #endregion

56 //第一个建造工厂

57 public class OneCreator : Creator

58 {

59 public override Product CreateProduct(string f\_Type)

60 {

61 switch (f\_Type)

62 {

63 case "one":

64 return new OneProduct();

65 case "two":

66 return new TwoProduct();

67 }

68

69 return null;

70 }

71 }

72 //第二个建造工厂

73 public class TwoCreator : Creator

74 {

75 public override Product CreateProduct(string f\_Type)

76 {

77 switch (f\_Type)

78 {

79 case "one":

80 return new FirstProduct();

81 case "two":

82 return new SecondProduct();

83 }

84 return null;

85 }

86 }

87

88

89

90 #endregion

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 static void Main(string[] args)

2 {

3

4

5 #region 工场模式

6

7

8

9 //第一个工厂 两种产品

10 Creator \_creator = new OneCreator();

11 Product \_product = \_creator.FactoryMethod("one");

12 Console.WriteLine(\_product.productName);

13 \_product = \_creator.FactoryMethod("two");

14 Console.WriteLine(\_product.productName);

15

16 //第二个工厂 造另两种产品

17

18 Creator \_tCreator = new TwoCreator();

19 Product \_tProduct = \_tCreator.FactoryMethod("one");

20 Console.WriteLine(\_tProduct.productName);

21 \_tProduct = \_tCreator.FactoryMethod("two");

22 Console.WriteLine(\_tProduct.productName);

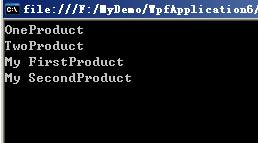
23 #endregion

24

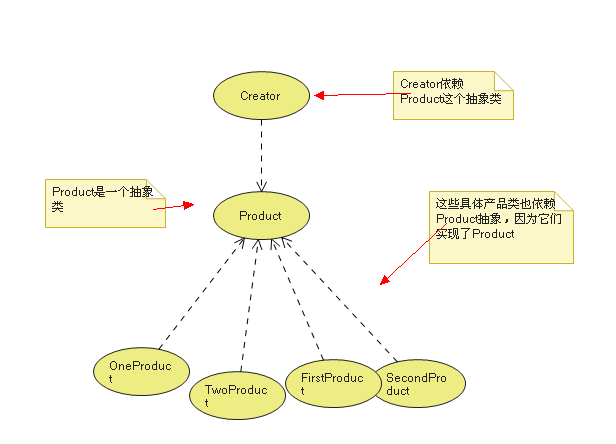
25 Console.ReadLine();

26 }

[复制代码](javascript:void(0);)



让我们来看一下依赖关系



我们会看到 Creator 和所有的产品（OneProduct、TwoProduct...）都依赖了Product.这是依赖倒置原则：要依赖抽象，不要依赖具体类

也就是说不能让具体产品去依赖Creator，不管是产品还是Creator都应该依赖于抽象

就用这个原则我们要尽量做到

1变量不可以持有具体类的引用（如果使用new就会有具体类的引用。你可以改用工厂来避开这样的做法）

2不要让类派生自具体类（派生自一个接口）

3不要覆盖基类中已实现的方法

但在实际编程时不可能完全遵守这几条，我们只要尽量做就可以了