**一、 模板方法（Template Method）模式**

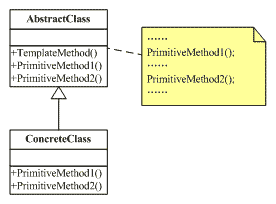
准备一个抽象类，将部分逻辑以具体方法以及具体构造子的形式实现，然后声明一些抽象方法来迫使子类实现剩余的逻辑。不同的子类可以以不同的方式实现这些抽象方法，从而对剩余的逻辑有不同的实现。这就是模版方法模式的用意。

很多人可能没有想到，模版方法模式实际上是所有模式中最为常见的几个模式之一，而且很多人可能使用过模版方法模式而没有意识到自己已经使用了这个模式。模版方法模式是基于继承的代码复用的基本技术，模版方法模式的结构和用法也是面向对象设计的核心。

模版方法模式需要开发抽象类和具体子类的设计师之间的协作。一个设计师负责给出一个算法的轮廓和骨架，另一些设计师则负责给出这个算法的各个逻辑步骤。代表这些具体逻辑步骤的方法称做基本方法（primitive method）；而将这些基本法方法总汇起来的方法叫做模版方法（template method），这个设计模式的名字就是从此而来。

**二、 模版方法模式的结构**

模版方法模式的静态结构如下图所示。



这里涉及到两个角色：

* 抽象模版（AbstractClass）角色有如下的责任：

定义了一个或多个抽象操作，以便让子类实现。这些抽象操作叫做基本操作，它们是一个顶级逻辑的组成步骤。

定义并实现了一个模版方法。这个模版方法一般是一个具体方法，它给出了一个顶级逻辑的骨架，而逻辑的组成步骤在相应的抽象操作中，推迟到子类实现。顶级逻辑也有可能调用一些具体方法。

* 具体模版（ConcreteClass）角色有如下的责任：

实现父类所定义的一个或多个抽象方法，它们是一个顶级逻辑的组成步骤。

每一个抽象模版角色都可以有任意多个具体模版角色与之对应，而每一个具体模版角色都可以给出这些抽象方法（也就是顶级逻辑的组成步骤）的不同实现，从而使得顶级逻辑的实现各不相同。

**三、 模板方法模式的示意性代码**

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Template Method pattern -- Structural example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "AbstractClass"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class AbstractClass  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void PrimitiveOperation1();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void PrimitiveOperation2();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // The Template method  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void TemplateMethod()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("In AbstractClass.TemplateMethod()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    PrimitiveOperation1();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    PrimitiveOperation2();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteClass"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass ConcreteClass : AbstractClass  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void PrimitiveOperation1()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Called ConcreteClass.PrimitiveOperation1()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void PrimitiveOperation2()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Console.WriteLine("Called ConcreteClass.PrimitiveOperation2()");  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// Client test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Client  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Create instance and call template method  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    ConcreteClass c = new ConcreteClass();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    c.TemplateMethod();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

**四、 继承作为复用的工具**

使用继承作为复用的手段必须慎重，C#语言的设计师对使用继承作为复用的工具有着不同层次上的认识。

不知其一

首先，初学C#的程序员可能不知道什么是继承，或者认为"继承"是高深的工具。那时候，大部分的功能复用都是通过委派进行的。

知其一、不知其二

然后慢慢地，他们发现在C#语言里实现继承并不困难，并且初步认识到继承可以使子类一下子得到基类的行为。这时他们就会跃跃欲试了，试图使用继承作为功能复用的主要工具，并把原来应当使用委派的地方，改为使用继承，这时继承就有被滥用的危险。

知其二

很多面向对象的设计专家从1986年就开始警告继承关系被滥用的可能。有一些面向对象的编程语言，如SELF语言，甚至将类的继承关系从语言的功能中取消掉，改为完全使用委派。

其他的设计师虽然不提倡彻底取消继承，但无一例外地鼓励在设计中尽可能使甩委派关系代替继承关系。比如在【GOF95】一书中，状态模式、策略模式、装饰模式、桥梁模式以及抽象工厂模式均是将依赖于继承的实现转换为基于对象的组合和聚合的实现，这些模式的要点就是使用委派关系代替继承关系。

知其三

是不是继承就根本不该使用呢？事实上对数据的抽象化、继承、封装和多态性并称C#和其他绝大多数的面向对象语言的几项最重要的特性。继承不应当被滥用，并不意味着继承根本就不该使用。因为继承容易被滥用就彻底抛弃继承，无异于因噎废食。

继承使得类型的等级结构易于理解、维护和扩展，而类型的等级结构非常适合于抽象化的设计、实现和复用。尽管【GOF95】所给出的设计模式基本上没有太多基于继承的模式，很多模式都是用继承的办法定义、实现接口的。多数的设计模式都描写一个以抽象类作为基类，以具体类作为实现的等级结构，比如适配器模式、合成模式、桥梁模式、状态模式等。

模版方法模式则更进了一步：此模式鼓励恰当地使用继承。此模式可以用来改写一些拥有相同功能的相关的类，将可复用的一般性的行为代码移到基类里面，而把特殊化的行为代码移到子类里面。

因此，熟悉模版方法模式便成为一个重新学习继承的好地方。

**五、 一个实际应用模板方法的例子**

下面的例子演示了数据库访问的模板方法。实际应用时，请确保C盘根目录下有nwind.mdb这个Access数据库（可以从Office的安装目录下找到。中文版用户的请注意字段名可能有所不同）。

http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// Template Method pattern -- Real World example    
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Data;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifusing System.Data.OleDb;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "AbstractClass"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifabstract class DataObject  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Connect();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Select();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Process();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  abstract public void Disconnect();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // The "Template Method"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public void Run()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Connect();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Select();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Process();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    Disconnect();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif// "ConcreteClass"  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifclass CustomerDataObject : DataObject  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string connectionString =  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    "provider=Microsoft.JET.OLEDB.4.0; "  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    + "data source=c:\\nwind.mdb";  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private string commandString;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  private DataSet dataSet;  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif   
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  // Methods  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Connect( )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Nothing to do  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Select( )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    commandString = "select CompanyName from Customers";  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    OleDbDataAdapter dataAdapter = new OleDbDataAdapter(  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      commandString, connectionString );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    dataSet = new DataSet();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    dataAdapter.Fill( dataSet, "Customers" );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Process()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    DataTable dataTable = dataSet.Tables["Customers"];  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    foreach( DataRow dataRow in dataTable.Rows )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif      Console.WriteLine( dataRow[ "CompanyName" ] );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public override void Disconnect()  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    // Nothing to do  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gif  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif///  TemplateMethodApp test  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/None.gifpublic class TemplateMethodApp  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif  public static void Main( string[] args )  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif  {  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    CustomerDataObject c = new CustomerDataObject( );  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/InBlock.gif    c.Run();  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif  }  
http://zhenyulu.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}

**六、 模版方法模式中的方法**

模版方法中的方法可以分为两大类：模版方法（Template Method）和基本方法（Primitive Method)。

模版方法

一个模版方法是定义在抽象类中的，把基本操作方法组合在一起形成一个总算法或一个总行为的方法。这个模版方法一般会在抽象类中定义，并由子类不加以修改地完全继承下来。

基本方法

基本方法又可以分为三种：抽象方法（Abstract Method）、具体方法（Concrete Method）和钩子方法（Hook Method）。

抽象方法：一个抽象方法由抽象类声明，由具体子类实现。在C#语言里一个抽象方法以abstract关键字标示出来。

具体方法：一个具体方法由抽象类声明并实现，而子类并不实现或置换。在C#语言里面，一个具体方法没有abstract关键字。

钩子方法：一个钩子方法由抽象类声明并实现，而子类会加以扩展。通常抽象类给出的实现是一个空实现，作为方法的默认实现。（Visual FoxPro中项目向导建立的项目会使用一个AppHook类实现监视项目成员变化，调整系统结构的工作。）钩子方法的名字通常以do开始。

**七、 重构的原则**

在对一个继承的等级结构做重构时，一个应当遵从的原则便是将行为尽量移动到结构的高端，而将状态尽量移动到结构的低端。

1995年，Auer曾在文献【AUER95】中指出：

1. 应当根据行为而不是状态定义一个类。也就是说，一个类的实现首先建立在行为的基础之上，而不是建立在状态的基础之上。
2. 在实现行为时，是用抽象状态而不是用具体状态。如果一个行为涉及到对象的状态时，使用间接的引用而不是直接的引用。换言之，应当使用取值方法而不是直接引用属性。
3. 给操作划分层次。一个类的行为应当放到一个小组核心方法（Kernel Methods）里面，这些方法可以很方便地在子类中加以置换。
4. 将状态属性的确认推迟到子类中。不要在抽象类中过早地声明属性变量，应将它们尽量地推迟到子类中去声明。在抽象超类中，如果需要状态属性的话，可以调用抽象的取值方法，而将抽象的取值方法的实现放到具体子类中。

如果能够遵从这样的原则，那么就可以在等级结构中将接口与实现分隔开来，将抽象与具体分割开来，从而保证代码可以最大限度地被复用。这个过程实际上是将设计师引导到模版方法模式上去。