**工厂方法模式（Factory Method）**

——.NET设计模式系列之五

Terrylee，2004年1月2日

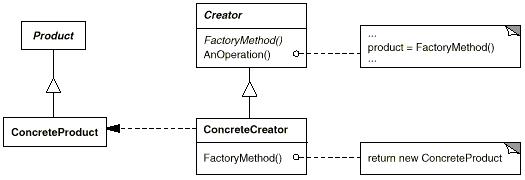
**概述**

在软件系统中，经常面临着“某个对象”的创建工作，由于需求的变化，这个对象的具体实现经常面临着剧烈的变化，但是它却拥有比较稳定的接口。如何应对这种变化？提供一种封装机制来隔离出“这个易变对象”的变化，从而保持系统中“其它依赖该对象的对象”不随着需求的改变而改变？这就是要说的Factory Method模式了。

**意图**

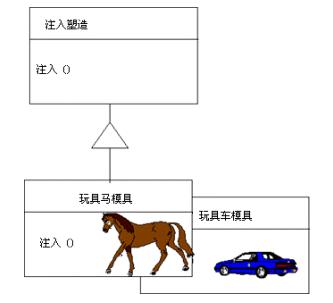
定义一个用户创建对象的接口，让子类决定实例化哪一个类。Factory Method使一个类的实例化延迟到其子类。

**结构图**



**生活中的例子**

工厂方法定义一个用于创建对象的接口，但是让子类决定实例化哪个类。压注成型演示了这种模式。塑料玩具制造商加工塑料粉，将塑料注入到希望形状的模具中。玩具的类别（车，人物等等）是由模具决定的。



**工厂方法解说**

在工厂方法模式中，核心的工厂类不再负责所有产品的创建，而是将具体创建工作交给子类去做。这个核心类仅仅负责给出具体工厂必须实现的接口，而不接触哪一个产品类被实例化这种细节。这使得工厂方法模式可以允许系统在不修改工厂角色的情况下引进新产品。在Factory Method模式中，工厂类与产品类往往具有平行的等级结构，它们之间一一对应。

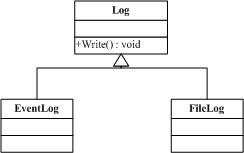
现在我们考虑一个日志记录的例子（这里我们只是为了说明Factory Method模式，实际项目中的日志记录不会这么去做，也要比这复杂一些）。假定我们要设计日志记录的类，支持记录的方法有FileLog和EventLog两种方式。在这里我们先不谈设计模式，那么这个日志记录的类就很好实现了：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// 日志记录类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class Log  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif    {  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public void WriteEvent()  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif        {  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            Console.WriteLine("EventLog Success!");  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public void WriteFile()  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif        {  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            Console.WriteLine("FileLog Success!");  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        public void Write(string LogType)  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif        {  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            switch(LogType.ToLower())  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif            {  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                case "event":  
22http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                    WriteEvent();  
23http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                    break;  
24http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
25http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                case "file":  
26http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                    WriteFile();  
27http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                    break;  
28http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
29http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                default:  
30http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                    break;  
31http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif            }  
32http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif        }  
33http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif    }  
34http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

这样的程序结构显然不能符合我们的要求，如果我们增加一种新的日志记录的方式DatabaseLog，那就要修改Log类，随着记录方式的变化，switch语句在不断的变化，这样就引起了整个应用程序的不稳定，进一步分析上面的代码，发现对于EventLog和FileLog是两种完全不同的记录方式，它们之间不应该存在必然的联系，而应该把它们分别作为单独的对象来对待。

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// EventLog类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class EventLog  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public void Write()  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Console.WriteLine("EventLog Write Success!");  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// FileLog类  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FileLog  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public void Write()  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Console.WriteLine("FileLog Write Success!");  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
22http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

进一步抽象，为它们抽象出一个共同的父类，结构图如下：



实现代码：

1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// Log类  
3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic abstract class Log  
5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public abstract void Write();  
7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
8http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

此时EventLog和FileLog类的代码应该如下：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// EventLog类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class EventLog:Log  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public override void Write()  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Console.WriteLine("EventLog Write Success!");  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// FileLog类  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FileLog:Log  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public override void Write()  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Console.WriteLine("FileLog Write Success!");  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

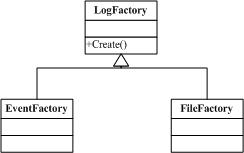
此时我们再看增加新的记录日志方式DatabaseLog的时候，需要做哪些事情？只需要增加一个继承父类Log的子类来实现，而无需再去修改EventLog和FileLog类，这样的设计满足了类之间的层次关系，又很好的符合了面向对象设计中的单一职责原则，每一个类都只负责一件具体的事情。到这里似乎我们的设计很完美了，事实上我们还没有看客户程序如何去调用。 在应用程序中，我们要使用某一种日志记录方式，也许会用到如下这样的语句：

http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifEventLog eventlog = new EventLog();  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifeventlog.Write();  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

当日志记录的方式从EventLog变化为FileLog，我们就得修改所有程序代码中出现上面语句的部分，这样的工作量是可想而知的。此时就需要解耦具体的日志记录方式和应用程序。这就要引入Factory Method模式了，每一个日志记录的对象就是工厂所生成的产品，既然有两种记录方式，那就需要两个不同的工厂去生产了，代码如下：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// EventFactory类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class EventFactory  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public EventLog Create()  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return new EventLog();  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// FileFactory类  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FileFactory  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public FileLog Create()  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return new FileLog();  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

这两个工厂和具体的产品之间是平行的结构，并一一对应，并在它们的基础上抽象出一个公用的接口，结构图如下：



实现代码如下：

1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// LogFactory类  
3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic abstract class LogFactory  
5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public abstract Log Create();  
7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
8http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

此时两个具体工厂的代码应该如下：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// EventFactory类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class EventFactory:LogFactory  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public override EventLog Create()  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return new EventLog();  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// FileFactory类  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class FileFactory:LogFactory  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public override FileLog Create()  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        return new FileLog();  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

这样通过工厂方法模式我们把上面那对象创建工作封装在了工厂中，此时我们似乎完成了整个Factory Method的过程。这样达到了我们应用程序和具体日志记录对象之间解耦的目的了吗？看一下此时客户端程序代码：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// App类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class App  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public static void Main(string[] args)  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        LogFactory factory = new EventFactory();  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Log log = factory.Create();  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        log.Write();  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

在客户程序中，我们有效地避免了具体产品对象和应用程序之间的耦合，可是我们也看到，增加了具体工厂对象和应用程序之间的耦合。那这样究竟带来什么好处呢？我们知道，在应用程序中，Log对象的创建是频繁的，在这里我们可以把

LogFactory factory = new EventFactory();

这句话放在一个类模块中，任何需要用到Log对象的地方仍然不变。要是换一种日志记录方式，只要修改一处为：

LogFactory factory = new FileFactory();

其余的任何地方我们都不需要去修改。有人会说那还是修改代码，其实在开发中我们很难避免修改，但是我们可以尽量做到只修改一处。

其实利用.NET的特性，我们可以避免这种不必要的修改。下面我们利用.NET中的反射机制来进一步修改我们的程序，这时就要用到配置文件了，如果我们想使用哪一种日志记录方式，则在相应的配置文件中设置如下：

1http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif<appSettings>  
2http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif    <add key="factoryName" value="EventFactory"></add>  
3http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif</appSettings>  
4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

此时客户端代码如下：

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif/// <summary>  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif/// App类  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif/// </summary>  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic class App  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif    public static void Main(string[] args)  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif    {  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        string strfactoryName = ConfigurationSettings.AppSettings["factoryName"];  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif          
10http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        LogFactory factory;  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        factory = (LogFactory)Assembly.Load("FactoryMethod").CreateInstance("FactoryMethod." + strfactoryName);  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        Log log = factory.Create();  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        log.Write();  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif    }  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

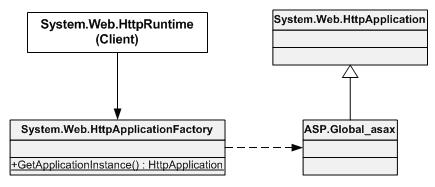
现在我们看到，在引进新产品（日志记录方式）的情况下，我们并不需要去修改工厂类，而只是增加新的产品类和新的工厂类（注意：这是任何时候都不能避免的），这样很好的符合了开放封闭原则。

**ASP.NET HTTP通道中的应用**

Factory Method模式在ASP.NET HTTP通道中我们可以找到很多的例子。ASP.NET HTTP通道是System.Web命名空间下的一个类，WEB Server使用该类处理接收到的HTTP请求，并给客户端发送响应。HTTP通道主要的工作有Session管理，应用程序池管理，缓存管理，安全等。

**System.Web.HttpApplicationFactory**

HttpRuntime是HTTP通道的入口点，它根据每一个具体的请求创建一个HttpContext实例， HttpRuntime并没有确定它将要处理请求的HttpApplication对象的类型，它调用了一个静态的工厂方法HttpApplicationFactory.GetApplicationInstance，通过它来创建HttpContext实例。GetApplicationInstance使用HttpContext实例来确定针对这个请求该响应哪个虚拟路径，如果这个虚拟路径以前请求过，HttpApplication（或者一个继承于ASP.Global\_asax的类的实例）将直接从应用程序池中返回，否则针对该虚拟路径将创建一个新的HttpApplication对象并返回。如下图所示：



HttpApplicationFactory.GetApplicationInstance带有一个类型为HttpContext的参数，创建的所有对象（产品）都是HttpApplication的类型，通过反编译，来看一下GetApplicationInstance的实现：

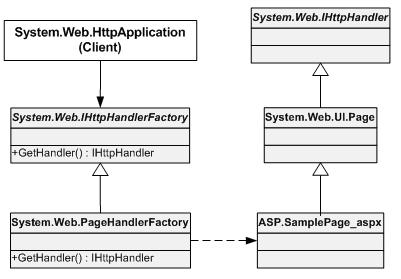
 1http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifinternal static IHttpHandler GetApplicationInstance(HttpContext context)  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      if (HttpApplicationFactory.\_customApplication != null)  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif      {  
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            return HttpApplicationFactory.\_customApplication;  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif      }  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      if (HttpDebugHandler.IsDebuggingRequest(context))  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif      {  
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            return new HttpDebugHandler();  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif      }  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      if (!HttpApplicationFactory.\_theApplicationFactory.\_inited)  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif      {  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif            lock (HttpApplicationFactory.\_theApplicationFactory)  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif            {  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                  if (!HttpApplicationFactory.\_theApplicationFactory.\_inited)  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif                  {  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                        HttpApplicationFactory.\_theApplicationFactory.Init(context);  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif                        HttpApplicationFactory.\_theApplicationFactory.\_inited = true;  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif                  }  
20http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif            }  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif      }  
22http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      return HttpApplicationFactory.\_theApplicationFactory.GetNormalApplicationInstance(context);  
23http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
24http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

**System.Web.IHttpHandlerFactory**

我们来做进一步的探索，HttpApplication实例需要一个Handler对象来处理资源请求， HttpApplication的主要任务就是找到真正处理请求的类。HttpApplication首先确定了一个创建Handler对象的工厂，来看一下在Machine.config文件中的配置区<httphandlers>，在配置文件注册了应用程序的具体处理类。例如在Machine.config中对\*.aspx的处理将映射到System.Web.UI.PageHandlerFactory 类，而对\*.ashx的处理将映射到System.Web.UI.SimpleHandlerFactory 类，这两个类都是继承于IhttpHandlerFactory接口的具体类：

http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif<httpHandlers>  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif<add verb="\*" path="\*.aspx" type="System.Web.UI.PageHandlerFactory" />  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif<add verb="\*" path="\*.ashx" type="System.Web.UI.SimpleHandlerFactory" />  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifhttp://www.cnblogs.com/Images/dot.gifhttp://www.cnblogs.com/Images/dot.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif</httpHandlers>  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

这个配置区建立了资源请求的类型和处理请求的类之间的一个映射集。如果一个.aspx页面发出了请求，将会调用System.Web.UI.PageHandlerFactory类，HttpApplication调用接口IHttpHandlerFactory中的工厂方法GetHandler来创建一个Handler对象。当一个名为sample.aspx的页面发出请求时，通过PageHandlerFactory将返回一个ASP.SamplePage\_aspx对象（具体产品），如下图：



IHttpHandlerFactory工厂：

1http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gifpublic interface IHttpHandlerFactory  
2http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gif{  
3http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      // Methods  
4http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      IHttpHandler GetHandler(HttpContext context, string requestType, string url, string pathTranslated);  
5http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      void ReleaseHandler(IHttpHandler handler);  
6http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}  
7http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

IHttpHandlerFactory.GetHandler是一个工厂方法模式的典型例子，在这个应用中，各个角色的设置如下：

抽象工厂角色：IHttpHandlerFactory

具体工厂角色：PageHandlerFactory

抽象产品角色：IHttpHandler

具体产品角色：ASP.SamplePage\_aspx

**进一步去理解**

理解上面所说的之后，我们就可以去自定义工厂类来对特定的资源类型进行处理。第一步我们需要创建两个类去分别实现IHttpHandlerFactory 和IHttpHandler这两个接口。

 1http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gifpublic class HttpHandlerFactoryImpl:IHttpHandlerFactory {  
 2http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif     
 3http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif   IHttpHandler IHttpHandlerFactory.GetHandler(  
 4http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      HttpContext context, String requestType,   
 5http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif      String url, String pathTranslated ) {  
 6http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
 7http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif         return new HttpHandlerImpl();  
 8http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif           
 9http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif   }//IHttpHandlerFactory.GetHandler  
10http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
11http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif   void IHttpHandlerFactory.ReleaseHandler(  
12http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif      IHttpHandler handler) { /\*no-op\*/ }  
13http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
14http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}//HttpHandlerFactoryImpl  
15http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
16http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockStart.gifpublic class HttpHandlerImpl:IHttpHandler {  
17http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
18http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif   void IHttpHandler.ProcessRequest(HttpContext context) {  
19http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        
20http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif      context.Response.Write("sample handler invokedhttp://www.cnblogs.com/Images/dot.gif");  
21http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif        
22http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockEnd.gif   }//ProcessRequest  
23http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
24http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedSubBlockStart.gif   bool IHttpHandler.IsReusable { get { return false; } }  
25http://terrylee.cnblogs.com/Images/OutliningIndicators/InBlock.gif  
26http://terrylee.cnblogs.com/Images/OutliningIndicators/ExpandedBlockEnd.gif}//HttpHandlerImpl  
27http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

第二步需要在配置文件中建立资源请求类型和处理程序之间的映射。我们希望当请求的类型为\*.sample时进入我们自定义的处理程序，如下：

http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif<httpHandlers>  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif   <add verb="\*" path="\*.sample"   
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif      type="HttpHandlerFactoryImpl,SampleHandler" />  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif</httpHandlers>  
http://terrylee.cnblogs.com/Images/OutliningIndicators/None.gif

最后一步我们需要把文件扩展\*.sample映射到ASP.NET ISAPI扩展DLL（aspnet\_isapi.dll）上。由于我们已经建立了用于处理新扩展文件的处理程序了，我们还需要把这个扩展名告诉IIS并把它映射到ASP.NET。如果你不执行这个步骤而试图访问\*.sample文件，IIS将简单地返回该文件而不是把它传递给ASP.NET运行时。其结果是该HTTP处理程序不会被调用。

运行Internet服务管理器，右键点击默认Web站点，选择属性，移动到主目录选项页，并点击配置按钮。应用程序配置对话框弹出来了。点击添加按钮并在可执行字段输入aspnet\_isapi.dll文件路径，在扩展字段输入.sample。其它字段不用处理；该对话框如下所示：



在.NET Framework中，关于工厂模式的使用有很多的例子，例如IEnumerable和IEnumerator就是一个Creator和一个Product；System.Security.Cryptography中关于加密算法的选择，SymmetricAlgorithm,AsymmetricAlgorithm, 和HashAlgorithm分别是三个工厂，他们各有一个静态的工厂方法Create；System.Net.WebRequest是 .NET Framework 的用于访问 Internet 数据的请求/响应模型的抽象基类。使用该请求/响应模型的应用程序可以用协议不可知的方式从 Internet 请求数据。在这种方式下，应用程序处理WebRequest 类的实例，而协议特定的子类则执行请求的具体细节。请求从应用程序发送到某个特定的 URI，如服务器上的 Web 页。URI 从一个为应用程序注册的 WebRequest 子代列表中确定要创建的适当子类。注册WebRequest 子代通常是为了处理某个特定的协议（如 HTTP 或 FTP），但是也可以注册它以处理对特定服务器或服务器上的路径的请求。有时间我会就.NET Framework中工厂模式的使用作一个专题总结。

**实现要点**

1．  Factory Method模式的两种情况：一是Creator类是一个抽象类且它不提供它所声明的工厂方法的实现；二是Creator是一个具体的类且它提供一个工厂方法的缺省实现。

2．  工厂方法是可以带参数的。

3．  工厂的作用并不仅仅只是创建一个对象，它还可以做对象的初始化，参数的设置等。

**效果**

1．  用工厂方法在一个类的内部创建对象通常比直接创建对象更灵活。

2．  Factory Method模式通过面向对象的手法，将所要创建的具体对象的创建工作延迟到了子类，从而提供了一种扩展的策略，较好的解决了这种紧耦合的关系。

**适用性**

在以下情况下，适用于工厂方法模式：

1.       当一个类不知道它所必须创建的对象的类的时候。

2.       当一个类希望由它的子类来指定它所创建的对象的时候。

3.       当类将创建对象的职责委托给多个帮助子类中的某一个，并且你希望将哪一个帮助子类是代理者这一信息局部化的时候。

**总结**

Factory Method模式是设计模式中应用最为广泛的模式，通过本文，相信读者已经对它有了一定的认识。然而我们要明确的是：在面向对象的编程中，对象的创建工作非常简单，对象的创建时机却很重要。Factory Method要解决的就是对象的创建时机问题，它提供了一种扩展的策略，很好地符合了开放封闭原则。\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

参考文献：

《设计模式》（中文版）

MSDN：《Exploring the Factory Design Pattern》

《DesignPatternsExplained》

支持TerryLee的创业产品[Worktile](https://worktile.com/?utm_source=cnblogs_terrylee)  
Worktile，新一代简单好用、体验极致的团队协同、项目管理工具，让你和你的团队随时随地一起工作。完全免费，现在就去了解一下吧。  
[https://worktile.com](https://worktile.com/?utm_source=cnblogs_terrylee)