第9章　面向复用的设计

9.1　软件复用的概念

我们可以把复用分为两类形式：组件复用和概念复用。

表9.1中所列的方法的使用取决于所要开发系统的需求，可复用技术和组件的有效性以及开发团队的经验。总的来说，进行软件复用时应考虑以下因素：

(1) 软件开发的进度。如果系统要求的进度较紧，复用时应采用大粒度组件的复用，尽量选择买来即可用(Off-the-shelf)的复用组件而不是采用独立(Individual)的复用组件。这样可以最大限度地减少开发工作量和风险。

(2) 期望的软件生命周期。如果系统要求生命周期长，则应关注系统的可维护性，复用时应考虑代码开源和具有稳定生命力的软件供应商。

表 9.1　支持软件复用的方法

方　法	描　述
设计模式(Design patterns)	对某种应用问题的解决方案的抽象
基于组件的开发(Component-based development)	通过集成遵从规范的一些组件完成系统的开发
应用框架(Application frameworks)	可被应用或扩展以建立新系统的具体或抽象的类
面向服务的系统(Service-oriented systems)	通过链接外部提供的可共享的服务建立新的系统
COTs 集成(COTs integration)	通过集成已有的应用系统
可配置的垂直的应用程序(Configurable vertical application)	设计一个通用的系统以便能通过配置适应特定的用户
程序库(Program libraries)	实现一些通用功能的类和函数库
程序生成器(Program generator)	可以针对特定类型的应用问题生成相应的程序或程序片段
面向方面的软件开发(Aspect-oriented development)	在程序编译时，将不同地方的可共享的组件编织(Weave)在一起

(3) 开发团队的技能和经验。复用技术通常比较复杂，要花费较大的代价理解和有效地应用。因此有经验的团队十分必要。

(4) 软件和其非功能性需求的重要程度。对于关键的系统，通常需要更高的可依赖性和性能，因此这类系统通常不赞成用生成器方法，那样会生成一些无效的代码。

(5) 应用域。有些应用域有着成熟的复用产品，比如制造业，只需配置这些产品来完成相应的需求。

(6) 系统运行的平台。一些复用组件适用于特定平台，如COM/Active X是Microsoft平台的产品，因此需要根据平台选择复用组件。

9.2　基于组件的开发

基于组件的软件工程是20世纪90年代后软件复用的重要方法，随着软件系统规模越来越大，复杂性也越来越高，虽然对象的封装特性可以完成一定程度上的复用，但由于对象类粒度过细，通常在编译时与具体的应用程序要进行绑定，因此需要对其有详细的了解方可使用，这样增加了复用的难度，使得单个的对象类组件没有市场化的价值。

9.2.1　组件

组件的特性主要包括：

(1) 规范性和通用性。

(2) 完整性和独立性。

(3) 可组合性。

(4) 可实施性。

(5) 灵活性。

(6) 文档特性。

9.2.2　组件模型

一个组件模型是有关组件实现、文档构建和组件部署(Deployment)等标准的定义。这些标准对组件开发者来说是保证组件互操作性的重要部分。典型的组件模型包括OMG的CORBA、Sun公司的EJB(Enterprise Java Bean)模型和Microsoft的COM+ 模型。

图9.1是Ian Somerville总结出的基本的组件模型，主要包括接口元素、在程序中使用组件的相关信息以及与组件实施相关的元素。

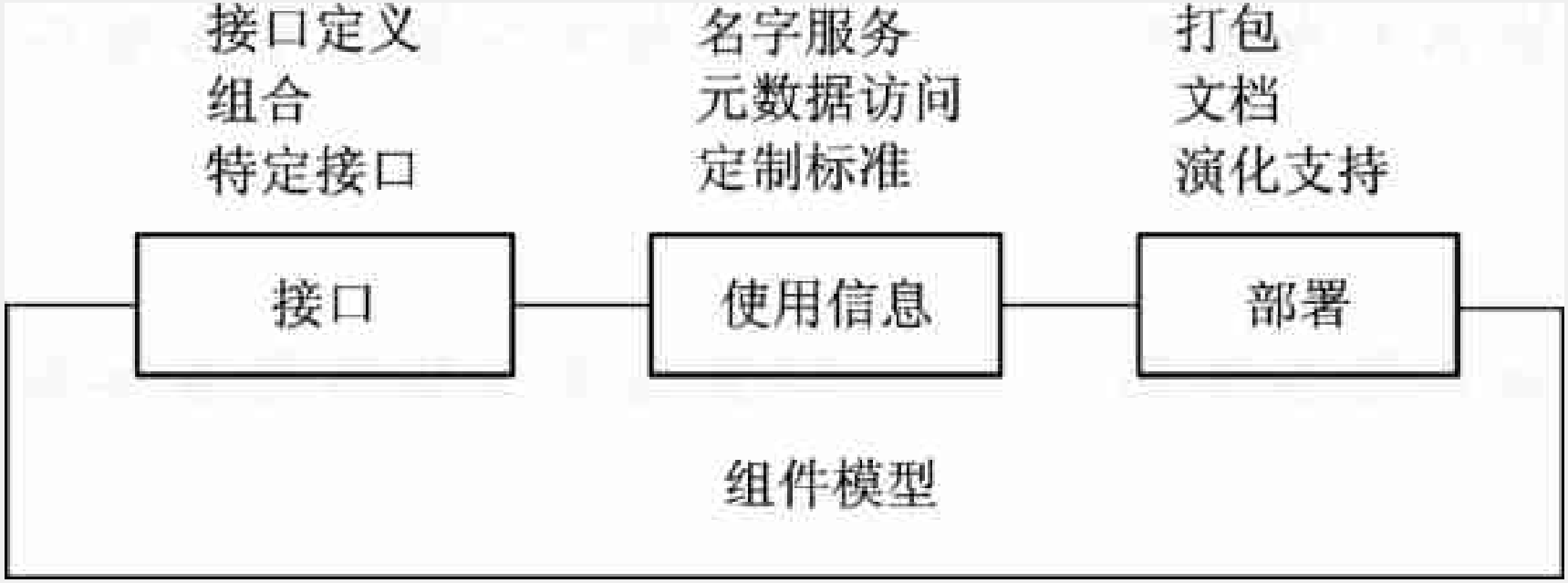

图9.1 基本的组件模型

9.2.3　中间件

中间件是一种类型的组件，在系统中通用的接口起到连接上层系统和下层系统的作用，故名中间件，如图9.2所示。中间件可以屏蔽下层异构、异质、分布式的环境，为上层系统提供统一的、透明的、简化的平台接口，以便于快速、方便地构造系统，是一种应用范围广泛的可复用技术。中间件主要分为过程中间件、对象中间件、事务中间件和消息中间件等几种类型。

图9.2 中间件的作用

9.2.4　基于组件的软件工程过程

基于组件的软件工程过程和其他的软件过程有一定的区别，主要包括需求定义、组件分析、需求的修正、基于复用的系统设计、组件集成等活动。基于组件的软件工程过程如图9.3所示。

图9.3 基于组件的软件工程过程

(1) 需求定义：初始的需求定义无须太详细。

(2) 组件分析：选择可信的组件库搜寻符合需求或接近需求的组件。

(3) 需求的修正：需求在过程早期可能因组件是否与需求吻合而做一定范围内的修改，这是由于组件是封装的成熟产品，不能更改，如果组件能满足需求的主体，而需求的定义有一定的柔性，适当修改需求以适应组件也是基于组件开发的重要环节。

(4) 基于复用的系统设计：在决定了所用的组件后，应根据组件和需求确定系统的体系结构，完成系统的设计。

(5) 组件集成：组件集成是组装组件构建系统的过程。

9.2.5　企业应用系统集成(EAI)

企业应用系统集成是一种对已有老(遗产)系统的复用技术。一个企业、机构的软件系统可能是其发展过程中不同阶段、不同需求的若干个相对独立的子系统，它们可能运行在不同的平台上，使用不同类型的数据库，有着不同的数据访问方式，同样的数据却存在着多个版本。

用户界面集成是一种最简单的集成，仅为已有系统通过集成的方式构建新的用户界面，如图9.4所示。已有的老系统在不改变其原有业务逻辑和表示逻辑的基础上，为系统建立新的、公共的用户界面。Web浏览器模式、Windows的GUI都是常用的统一界面风格。用户界面的集成是低风险和低代价的，技术成熟且容易达到目标。但仅仅实现了统一用户界面，并非是系统的实质性整合。

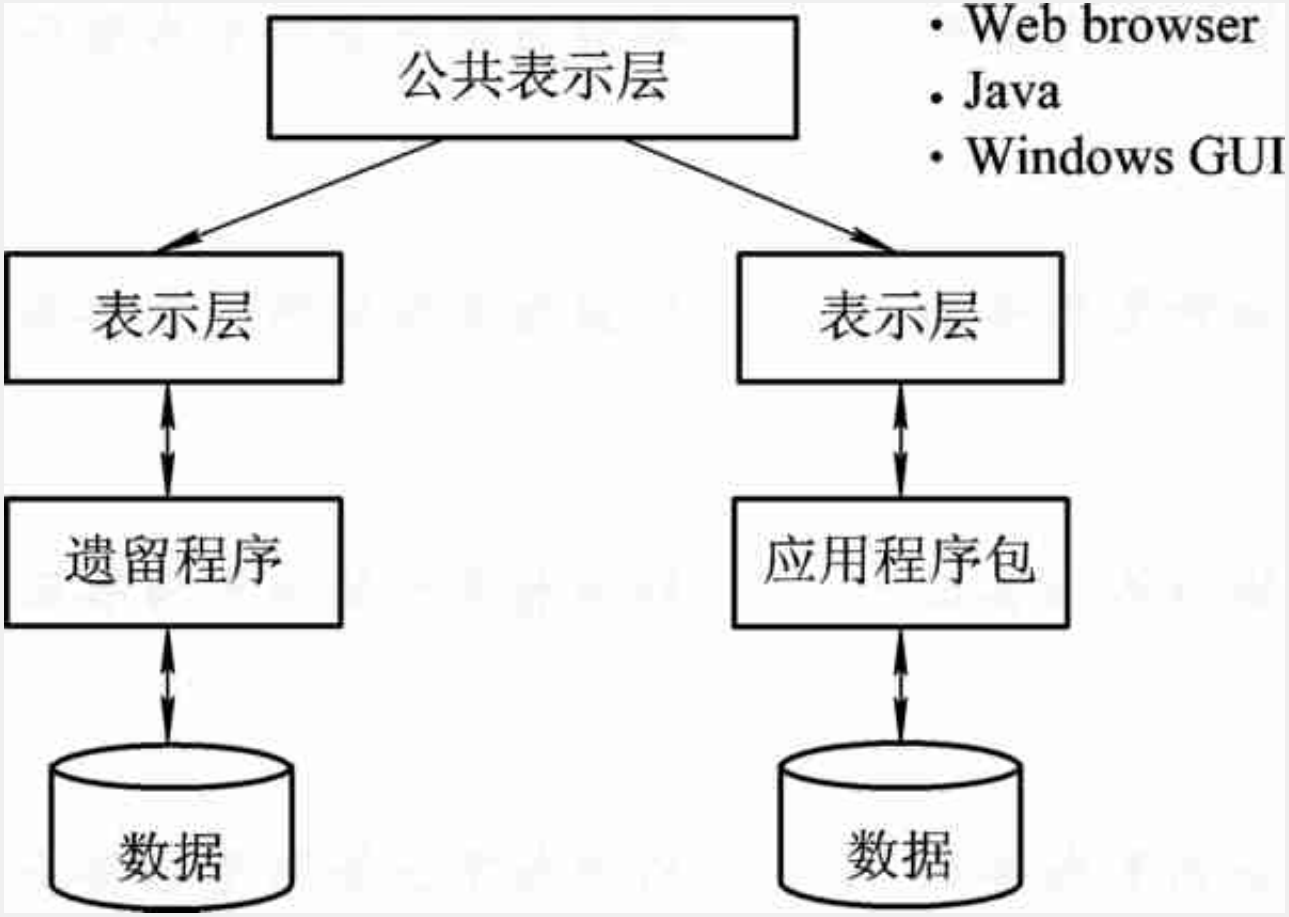

图9.4 用户界面集成

数据集成则是完成若干个应用系统之间数据的统一存储、管理和访问机制的集成。数据集成分析来自多个数据源的数据，形成统一的数据存储访问策略，如图9.5所示。数据集成可以通过数据访问中间件或一些工具来完成，如ODBC(Open Database Connectivity)、JDBC、数据仓库工具ETL(Extract, Transform and Load)。

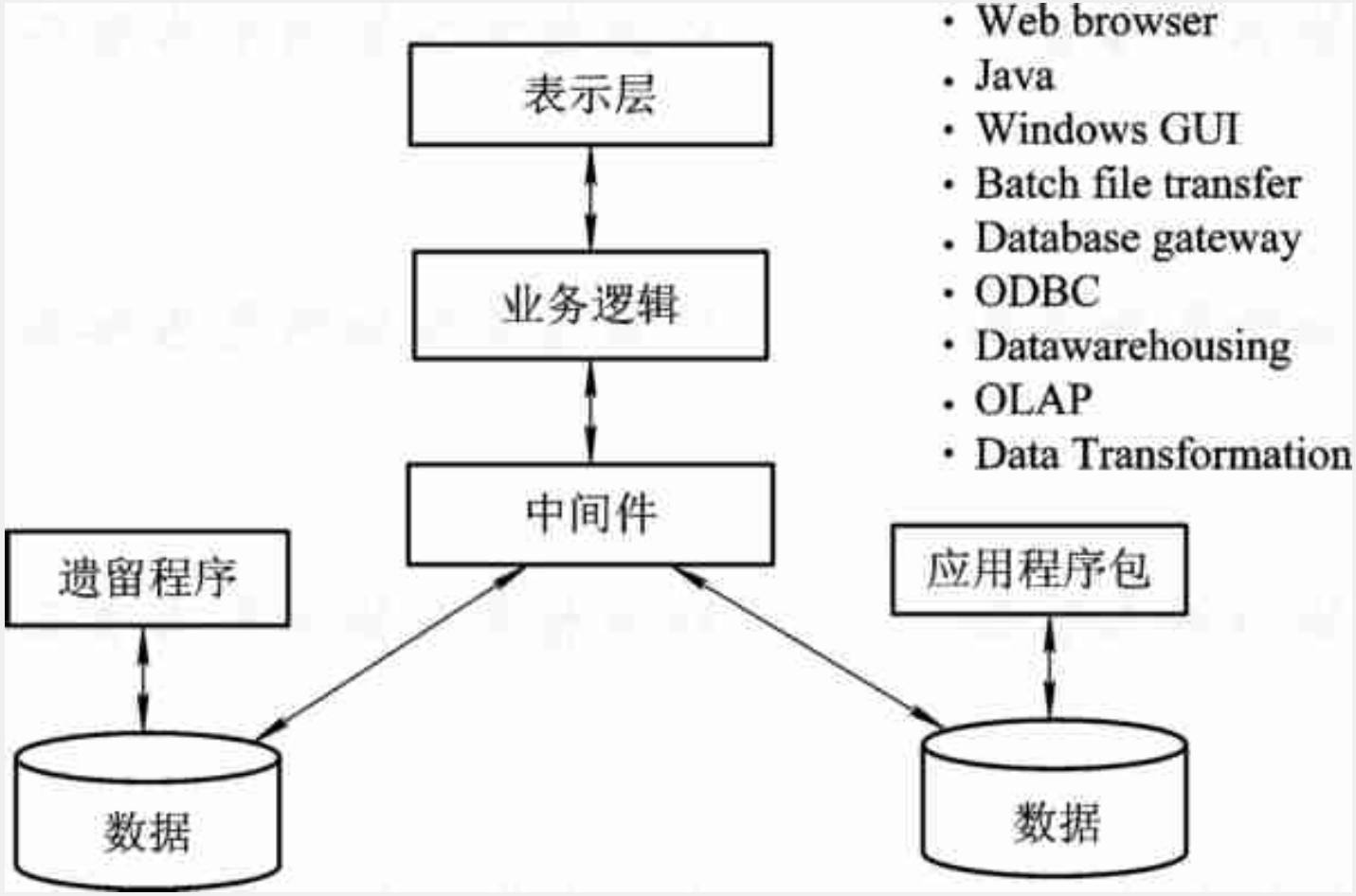
表示层
业务逻辑
中间件
遗留程序
应用程序包
数据
数据
• Web browser
• Java
• Windows GUI
• Batch file transfer
• Database gateway
• ODBC
• Datawarehousing
• OLAP
• Data Transformation

图9.5 数据集成

功能集成则建立应用系统之间功能的相互调用。功能集成通常使用分布式处理中间件完成各个系统业务逻辑的集成，如图9.6所示。分布式处理中间件可以提供软件系统组件之间的对象请求和消息通信，用于集成的中间件主要包括消息中间件MOM (Message Oriented Middleware)、分布式对象中间件DOT(Distributed Object Technology)和事务处理中间件TPMs(Transaction Processing Monitors)。

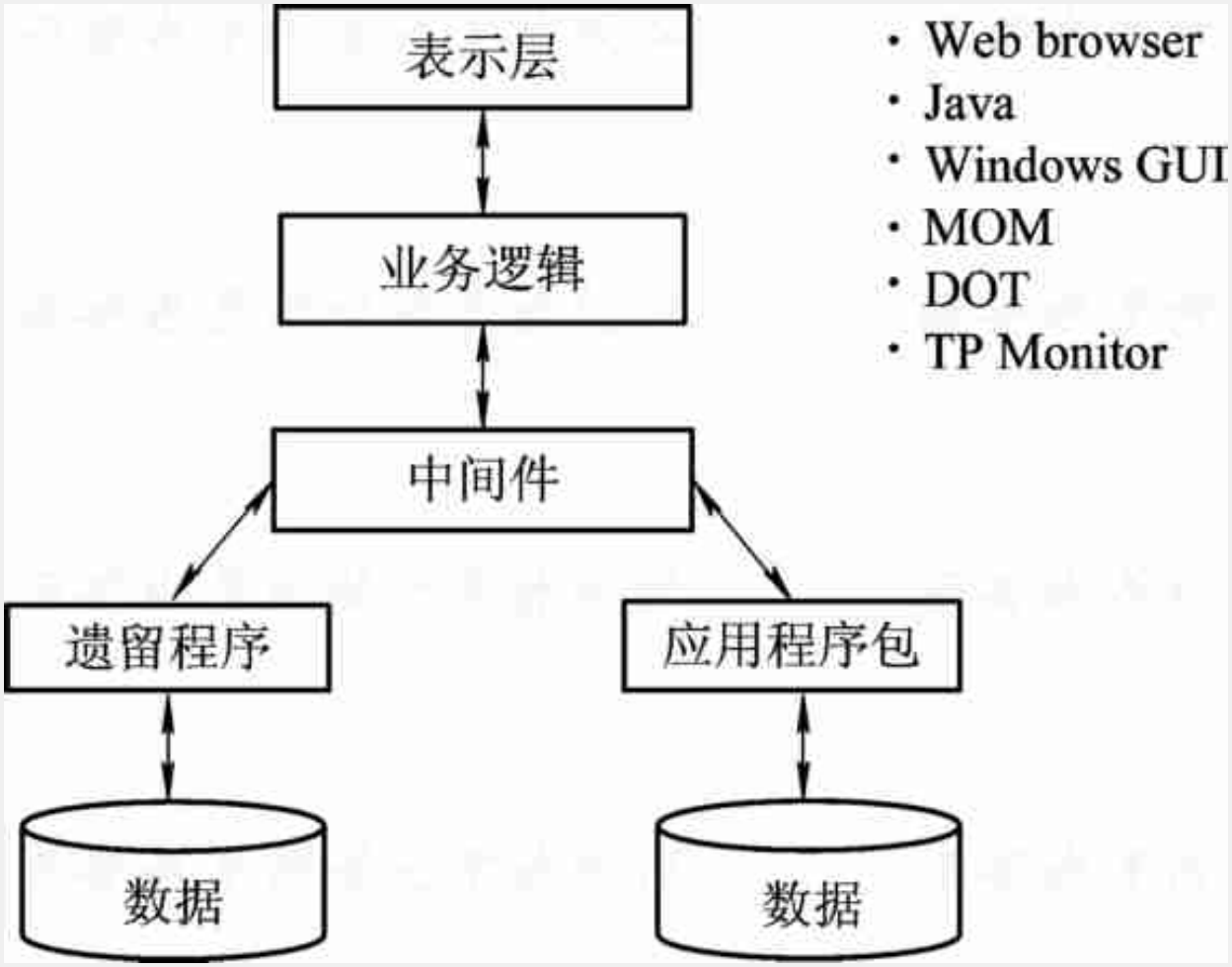
表示层
业务逻辑
中间件
遗留程序
应用程序包
数据
数据
· Web browser
· Java
· Windows GUI
· MOM
· DOT
· TP Monitor

图9.6 功能集成

功能集成是最具鲁棒性的集成，是企业应用系统真正的代码复用，是企业应用系统的真正解决方案。但功能集成是三种集成方式中最复杂、代价最高的。

表9.2列出了Sun公司提供构建和集成企业应用系统的组件和软件技术。

表 9.2　Sun 公司提供构建和集成企业应用系统的组件和软件技术

组　件	应　用
Java Applets, Java Servlets,JSP	Web 客户方
JDBC	数据库访问
Enterprise JavaBean(EJB)	分布式组件服务
Remote Method Invocation(RMI)	分布式对象服务
Java Naming and Directory(JNDI)	命名服务
Java Transaction Services(JTS)	事务服务
Java Messaging Services(JMS)	消息服务

9.3 设 计 模 式

9.3.1 设计模式概念

设计模式主要支持面向对象的复用，模式经常依赖于对象的继承、多态等特征。Gamma等四人的《设计模式》一书第一次将设计模式提升到理论高度，并将之规范化。书中提出了23种基本设计模式，自此，在可复用面向对象软件的发展过程中，新的大量的设计模式不断出现。Gamma等人定义了设计模式的四个基本元素：模式名称(Pattern Name)、问题域(Problem)、解决方案(Solution)以及效果(Consequence)。

设计模式在粒度和抽象层次上各不相同，根据其使用目的，设计模式可以分成三类：创建模式、结构模式和行为模式。创建模式与类和对象的创建有关；结构模式处理类和对象的组合；行为模式描述类或对象间的交互和职责分配，关注对象间如何协作完成工作。根据其作用于类或对象，设计模式又分为类模式和对象模式。类模式主要是指处理类之间关系的模式。表9.3列出了23种经典的设计模式，表9.4则给出了这些模式的分类情况。

表 9.3　经典的设计模式

模　式	描　述
Abstract Factory	提供一个创建一系列相关或相互依赖对象的接口，而无需指定它们具体的类
Adapter	将一个类的接口转换成客户希望的另外一个接口。Adapter 模式使得原本由于接口不兼容而不能一起工作的类可以一起工作
Bridge	将抽象部分与它的实现部分分离，使它们都可以独立地变化
Builder	将一个复杂对象的构建与它的表示分离，使得同样的构建过程可以创建不同的表示
Chain of Responsibility	消除请求的发送者和接收者之间的耦合，将对象连成一条链，并沿着这条链传递该请求，直到有一个对象处理它，使多个对象都有机会处理这个请求
Command	将一个请求封装为一个对象，从而可用不同的请求对客户进行参数化；对请求排队或记录请求日志，支持可取消的操作
Composite	将对象组合成树形结构以表示“部分-整体”的层次结构。使客户对单个对象和复合对象的使用具有一致性
Decorator	可动态地给一个对象添加职责。对扩展功能而言，　比生成子类方式更为灵活
Façade	Facade 模式定义了一个高层接口，为子系统中的一组接口提供一致的界面，使得子系统更加容易使用
Factory Method	定义一个用于创建对象的接口，让子类决定将哪一个类实例化。Factory Method 使一个类的实例化延迟到其子类
Flyweight	运用共享技术有效地支持大量细粒度的对象
Interpreter	给定一个语言，定义它的文法的一种表示，并定义一个解释器，该解释器使用该表示来解释语言中的句子

续表

模　式	描　述
Iterator	提供一种方法顺序访问一个聚合对象中各个元素，而无须知道该对象的内部表示
Mediator	用一个中介对象来封装一系列的对象交互。中介者使各对象不需要显式地相互引用，从而使它们具有松散的耦合，且可以独立地改变它们之间的交互
Memento	在不破坏封装性的前提下，捕获一个对象的内部状态，并在该对象之外保存这个状态，以便今后可将该对象恢复到保存的状态
Observer	定义对象间的一种一对多的依赖关系,以便当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动刷新
Prototype	用原型实例指定创建对象的种类，并且通过拷贝这个原型来创建新的对象
Proxy	为其他对象提供一个代理以控制对这个对象的访问
Singleton	保证一个类仅有一个实例，并提供一个访问它的全局访问点
State	允许一个对象在其内部状态改变时改变它的行为
Strategy	定义一系列的算法，对它们进行封装，并使它们可相互替换。本模式使得算法的变化可独立于使用它的客户
Template Method	定义一个操作中的算法的骨架，而将一些步骤延迟到子类中。Template Method使得子类可以不改变一个算法的结构即可重定义该算法的某些特定步骤
Visitor	表示一个作用于某对象结构中的各元素的操作，可以在不改变各元素的类的前提下定义作用于这些元素的新操作

表 9.4　模式的分类

	创建模式	结构模式	行为模式
类	Factory Method	Adapter	Interpreter Template Method
对象	Abstract Factory Builder Prototype Singleton	Adapter Bridge Composite Decorator Façade Flyweight Proxy	Chain of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

9.3.2 Composite模式

一种好的解决方案是定义一个抽象类Graphic，如图9.7所示，它既可以代表图元类也可以表示容器类。Graphic类定义了基本图形和组合图形类一般都具有的通用操作接口draw()、add()、remove()等,也定义了组合图形类特有的操作getChild()。Graphic的子类Line、Rectangle、Text是具体实现图元的类，而Picture子类则定义了一个组合图形类的具体实现，它是图形对象的聚合。Picture的接口和Graphic的接口是一致的，从图9.8可以看出Picture对象可以递归组合其他Picture对象。

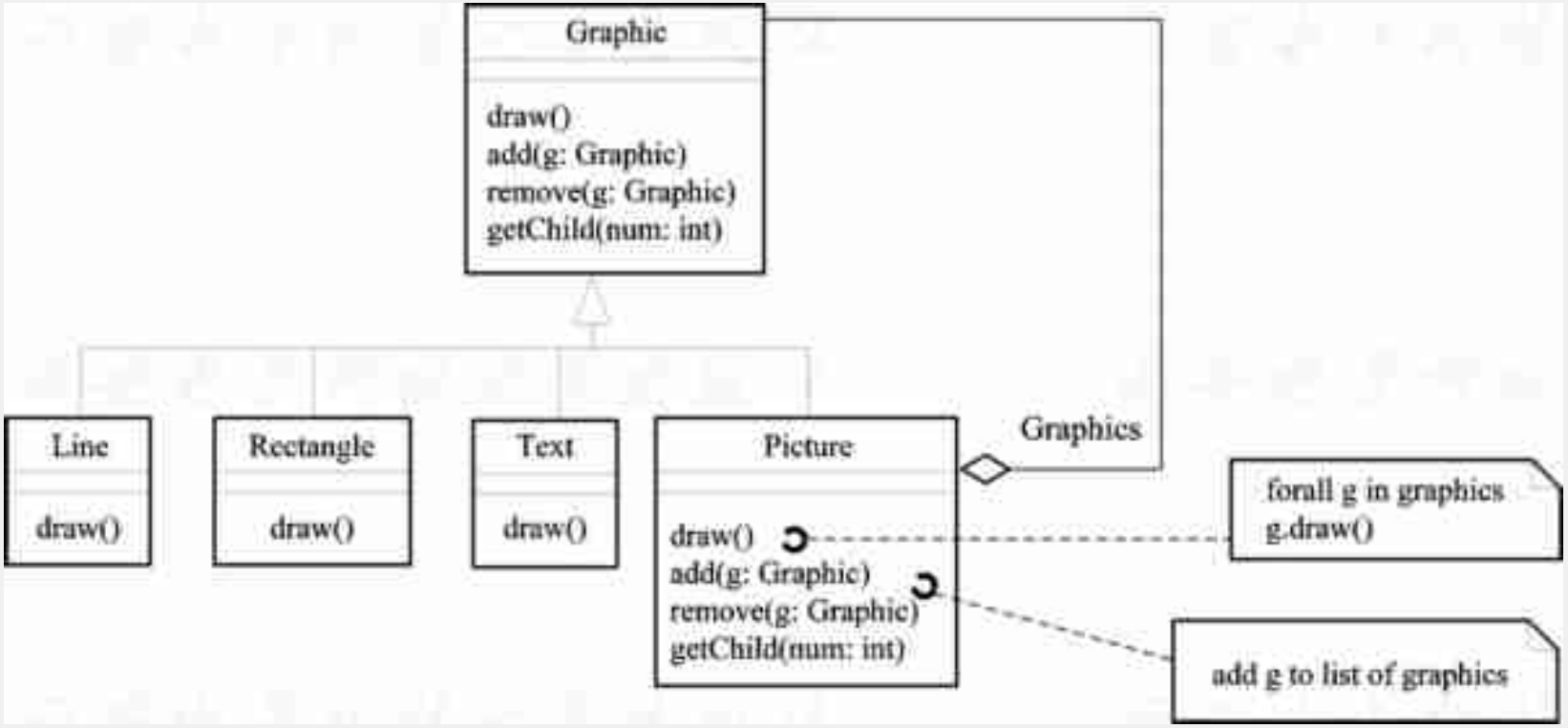

图9.7 Composite模式示例