

# Introduction to Application Frameworks

## 第 1 章

# 应用框架介绍

作为本书的作者，我的目的只有一个：使你相信应用框架（application framework）的价值。贯穿全书，我们将讨论应用框架的许多方面。例如，我将说明为什么框架在应用开发中很重要，以及什么技术能用于开发应用框架。我推崇通过具体实例讲解的方式，因此，我将通过一个我开发的、名为 SAF(Simplified Application Framework)的应用框架作为范例，讲解如何开发应用框架。

作为范例的应用框架 SAF 是用 C# 编写的。C# 是一种计算机编程语言，它能够为应用框架的开发提供面向对象特征。如果具有良好的 Microsoft .NET 框架、C# 和面向对象编程实践经验，无疑将对理解本书涉及的相关技术大有裨益。.NET 技术的很多特性可能是你已经知道、但还没有亲自用过的高级主题，我们在讨论应用框架的时候将涉及到它们；你也可以借此机会看看在应用框架 SAF 中，有多少这样的特性被实际实现和使用。本书也涉及到了“四人组”（gang of four, GOF）设计模式，从中你能学到大量优秀应用设计技术，应用框架经常依靠这些模式来达到设计重用(reuse)的目的。应用框架 SAF 作为范例贯穿全书，当我讲解其中的众多 GOF 设计模式是如何实现的时候，你会学到更多。读过本书之后，如果你接受了应用框架的思想，同时还学到了许多.NET 技术和设计模式，并能将它们用在你的项目中，作为本书的作者，我将感到非常高兴。

## 1.1 什么是应用框架

在说服你接受应用框架的思想之前，我们首先需要来定义应用框架是什么。我们从美国传统词典（American Heritage Dictionary）中对框架（framework）的定义开始吧。“支撑或围住其他物体的结构，尤指用作建筑物之基础的支撑骨架；一种基本结构，如关于一部作品或一系列观点的基本结构（A structure for supporting or enclosing something else, especially a skeletal support used as the basis for something being constructed; a fundamental structure, as for a written work or a system of ideas）。”

术语“框架”对不同的人，含义不同。政治家用这个词描述某些政策和解决问题的某些措施。建筑师用这个词描述建筑物的骨架或结构。软件架构师用这个词描述有助于软件应用开发的一组可重用的设计和代码。“框架”的最后这个含义，正是本书其余部分要讨论的。

“结构（structure）”一词着实反映了任何框架的本质。结构无处不在。当开始建造一座新的建筑物时，你会观察到，它的结构先被搭建。当我写书的时候，我也首先为书中要讨论的内容拟订一个结构，或者称为提纲。通过搭建一个“结构”，可以迫使我看全局。对建筑来说，着眼全局将迫使建筑师深入考虑：建筑物的每一部分会如何影响其他每一部分。对写书来说，着眼全局会迫使我去考虑如何组织每一章和每一个主题，才能使整本书易于读者理解。

结构在应用开发中也扮演了重要角色。一个相当复杂的应用包含了如此之多不断变化的细枝部分，以致人们无法把握它们的复杂关系。而结构帮助我们将这些不断变化的细枝部分，组织成易于理解的少数几个主要部分。在开始开发应用时，结构能为我们提供进一步实现的上下文（context）。应用框架为开发者提供了结构和模板，开发者以此为基线（baseline）来构建他们的应用系统。这样一个框架通常会包含抽象类（abstract classes）、具体类（concrete classes）和类之间的预定义交互（predefined interaction）。开发者在框架之上构建应用，通过重用由框架提供的代码和设计，减小了开发工作量。图 1-1 概括了应用框架和业务应用（business application）的关系。

当然，许多应用系统的开发没有使用框架，所以，你也许做着开发工作却甚至并不知道框架的概念。在应用开发领域，无论有没有框架，所有事情照样能做。然而，框架能为应用提供很多好处，采用应用框架方法对应用开发大有裨益。而推广应用框架的主旨，就是为了获得这些好处。

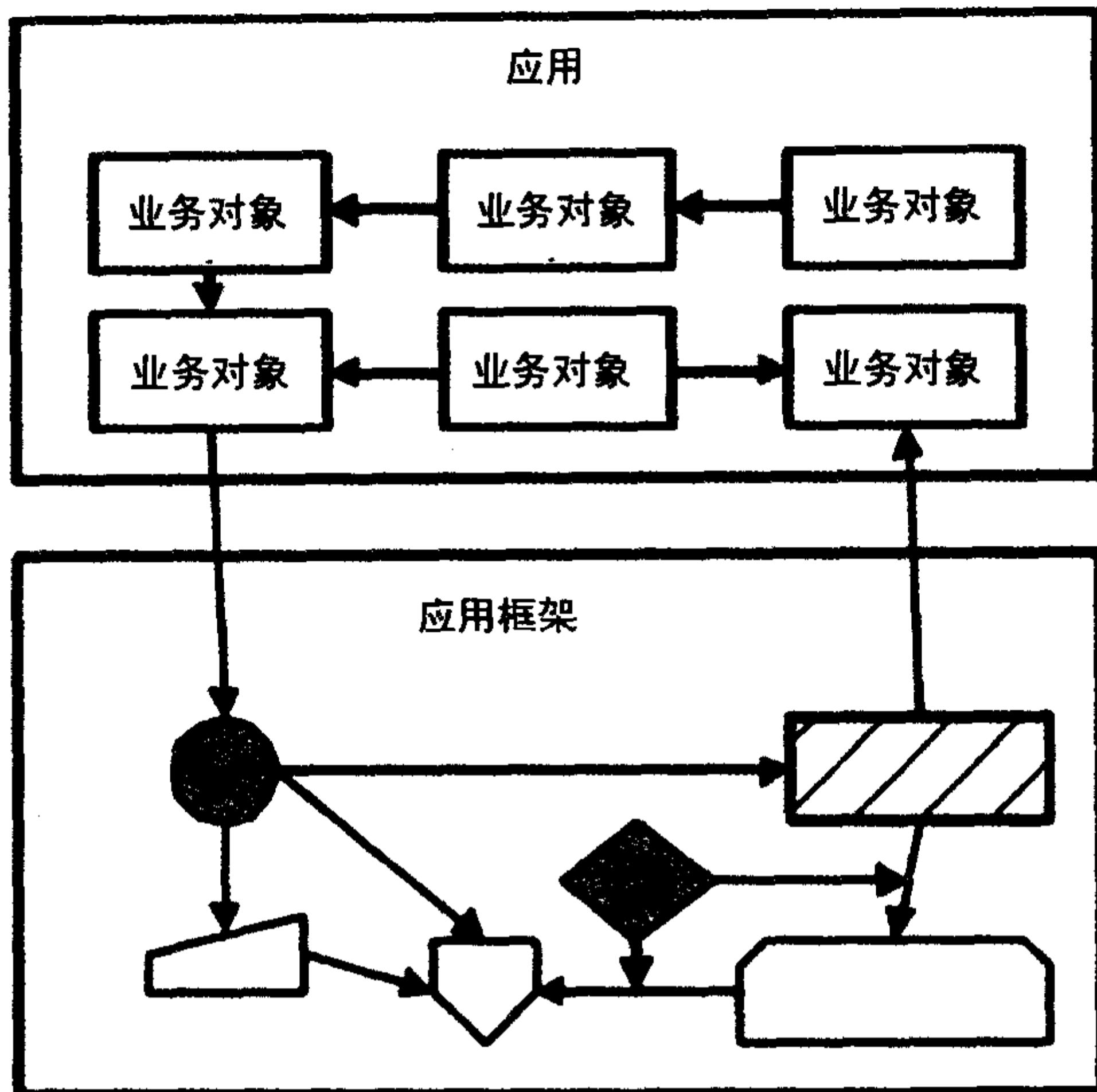


图 1-1 应用和应用框架的关系概括

## 1.2 应用框架的历史

应用框架并不是新概念，各种类型的框架已经被使用大约二十年了。第一个被广泛使用的框架是模型-视图控制器（Model-View Controller, MVC），它是一个由施乐（Xerox）公司开发的 Smalltalk 用户界面框架。这种使用观察者设计模式（Observer design pattern）的 MVC 方法已经被很多用户界面系统采用。除了 Smalltalk MVC 以外，还涌现出其他一些用户界面框架，它们对开发运行于不同操作系统之上的应用都起到了辅助作用。著名的用户界面框架有 MacApp 和 MFC，它们分别辅助 Macintosh 和 Windows 操作系统之上的应用开发。

虽然框架概念在用户界面开发中被广泛采用，但是它并不局限于用户界面框架。框架概念也用于通用应用开发。Taligent 是一家开发面向对象操作系统的公司，它曾提醒软件业界注意框架概念的潜力。Taligent 公司在 1992 年由 IBM 和 Apple 公司合作创建，目的是开发可以运行在任何硬件平台之上的新型操作系统。然而，随着时间

的推移，业界对这种新型操作系统的兴趣降低了。于是，Taligent 公司就转向开发基于现有操作系统的框架了。CommonPoint 是 Taligent 公司开发的一个框架，旨在减小应用开发的工作量，其方法是为开发者提供一个综合编程环境（comprehensive programming environment），类似于 Sun 公司如今提供的包括 Java 语言和运行时虚拟机的 Java 环境。

Sun 公司的 Java 环境和 Microsoft 公司的.NET 环境，不仅提供了新型的语言和虚拟机，还提供了它们自己的框架。在过去几年中，使用 Java 或.NET 的开发者能够充分感觉到这两种框架为他们的应用开发所带来的好处。Java 和.NET 都是旨在支持所有业务类型的应用系统的通用框架，因此，它们不能包含任何业务领域相关的类和设计。然而，基于这些通用框架之上的业务框架，可以为诸如供应链系统和金融应用等特定业务领域提供相关服务和专门支持。

IBM（后来收购了 Taligent）也开发了自己的面向业务领域的框架，被称为 San Francisco 项目。它是用 Java 开发的，由针对不同业务领域的应用框架组成，例如，订单管理、仓库管理、财务会计核算（general ledger management）等。与 Java 和.NET 这些通用框架不同，San Francisco 框架是专门为特定业务领域而设计的。

## 1.3 为何使用应用框架

使用应用框架有五大优点：模块化（modularity）、可重用性（reusability）、可扩展性（extensibility）、简单性（simplicity）和可维护性（maintainability）。

### 1.3.1 模块化

模块化就是把应用分割成多个组件或模块。这样，开发者就可以采用各模块互不影响的方式使用应用框架——希望使用应用框架某个组件的开发者，不会受到框架其他组件潜在变化的影响。当开发者基于框架之上构建应用时，他们的开发活动会更好地被隔离开，而不受应用框架其他部分变化的影响；从而，开发效率显著提高了。把框架划分成模块，我们就可以指派擅长该部分的开发者来完成它，从而使生产效率最大化。以开

以开发 Web 应用为例，模块化带来如下好处：指派熟悉表现层的用户界面开发者负责应用的前端（front-end），开发效率会更高；而指派熟悉业务逻辑层的开发者负责应用的中间层（middle tier）和后端（back-end），开发效率也会更高。同样，对使用应用框架来说，有的开发者擅长使用框架的用户界面相关模块，而有的开发者擅长使用框架的业务对象。

### 1.3.2 可重用性

代码的可重用性是应用开发中最重要和最令人期待的目标。应用框架能为基于其上构建的应用提供这种可重用性；而且，不仅应用框架的类和代码被重用了，其设计也被重用了。不同的应用通常会包含很多本质上相似的任务，然而，团队中的不同开发者往往都会自己实现一遍。这种重复实现，不仅在重复的代码上不必要地浪费了资源，也为以后的维护带来了困难。这是因为，要保证修改得彻底，必须对应用的多处进行重复修改。另外，由于每个开发者的设计方法可能不尽相同，这就可能使应用的设计变得很糟糕，为以后带来不可预料的问题。然而，有了应用框架，我们能将大量重复代码和通用解决方案从应用层移到框架层。这样一来，开发者编写和维护的重复代码的数量减少了，开发效率也大幅提高了。应用框架是精心设计的组件的汇集之处，其中不乏“久经考验”的优秀的软件设计方案。开发者并不一定是软件设计专家，但是一旦开始使用框架组件来构建应用，他们就自然而然地在重用优秀的软件设计方法了，比如藏在框架组件背后的设计模式等等。

### 1.3.3 可扩展性

可扩展性是往现有的框架中增加自定义功能的能力，它使开发者不仅能够“即拆即用（out of the box）”地使用框架组件，还能够改变组件，以适应特定业务场景（business scenario）的需要。可扩展性是框架的重要特征。每一个业务应用都有独一无二的业务需求、架构（architecture）和实现。虽然框架本身容纳所有这些具体情况是不可能的，但

是，框架采取了支持客户化（customization）的设计思路；这样，不同的业务应用依然能够使用框架的通用功能，同时，开发者通过在框架中插入自定义的业务逻辑，可以自由地按照独一无二的业务需求剪裁他们的应用。框架本身有了很高的可扩展性，就能适应更多类型的业务应用。然而，在创建框架时，其可扩展性总应根据它将支持的应用的设计和上下文来决定。框架的可扩展性越高，使用框架的开发者需要编写的代码就越多，他们需要了解的框架机制细节也越多，这样就会降低开发效率。一个高度可扩展框架的极端例子是 Microsoft .NET 框架，它所支持的应用范围非常广泛。的确，使用.NET 框架开发应用几乎没有限制，但结果，你也失去了应用框架能够提供的好处。关键在于，找出你要开发的特定应用中最可能发生变化之处，在那里增加灵活性（flexibility）和可扩展性支持。

### 1.3.4 简单性

术语“简单性（simplicity）”的含义不仅仅是“简单”。简单性指的是一种方法：框架通过封装处理流程的控制逻辑，使它对开发者透明，来简化开发工作。这种封装也是框架和类库（class library）的区别之一。类库由许多现成的、供开发者用于构建应用的组件组成，但是，开发者必须理解不同组件之间的关系，并编写处理流程代码把众多组件组织起来。框架则不同，它通过预先把众多组件组织在一起的方式，封装了处理流程的控制逻辑；因此，开发者就不用再编写控制逻辑来组织组件之间的交互了。

图 1-2 说明了类库和框架的这种区别。

由图可知，应用开发者使用类库这种方法时，必须编写管理类库中不同组件实例（instance）的控制流程。为此，应用开发者必须充分理解每个相关组件，以及组织组件协作所必需的业务逻辑。而使用框架这种方法时，由于大部分处理流程已经被框架管理了起来，所以开发者需要编写的控制代码就非常少。由于应用框架隐藏了不同组

件之间的处理流程，这就免去了开发者编写协调逻辑（coordination logic）之苦，也不用经历编写这些协调代码的学习曲线了。既然处理流程的控制逻辑从应用层移到了应用框架层，那么框架的设计人员就要运用其架构和领域知识，来定义框架内的组件该如何协作；而使用框架的开发者，几乎无须知道框架组件如何协作，就能高效地开发应用。

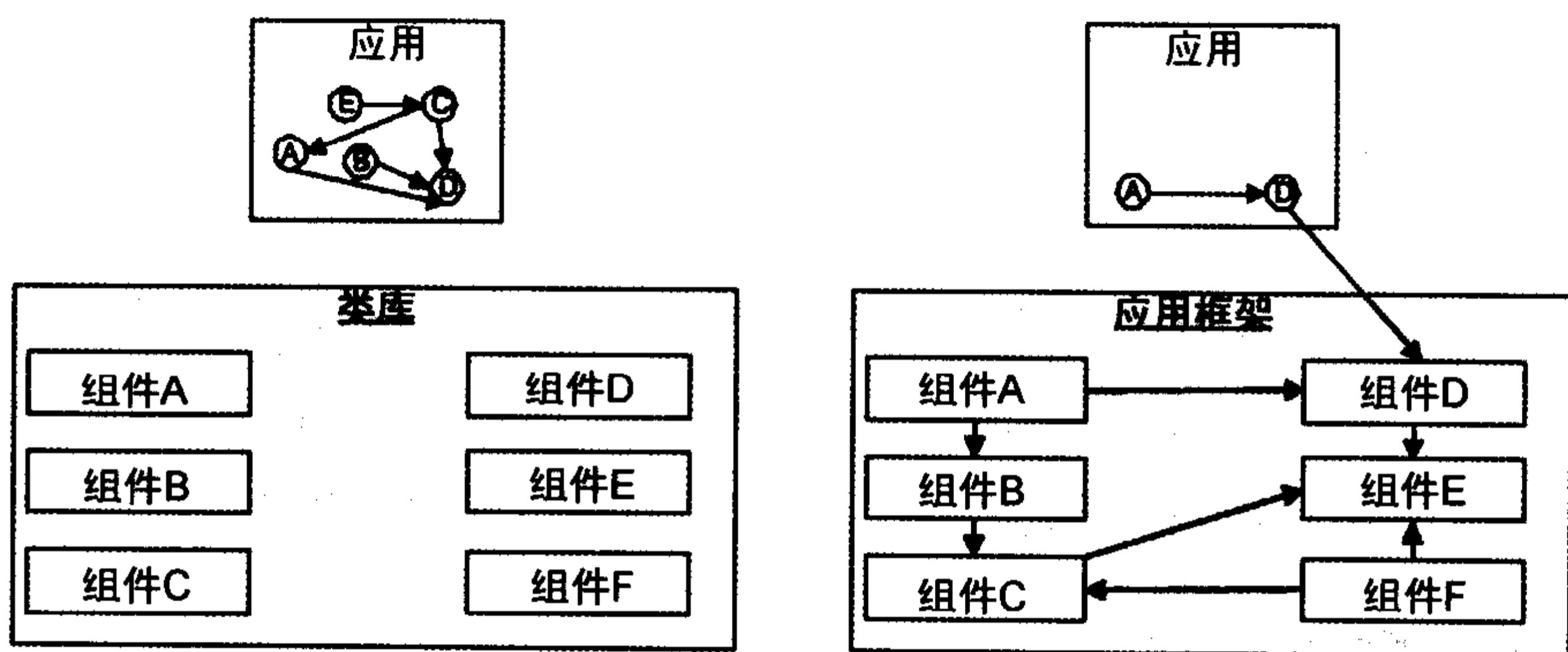


图 1-2 类库和应用框架的比较

### 1.3.5 可维护性

可维护性是业务需求改变之后，其应用便于修改的能力。它是代码重用带来的一个受欢迎的效用。框架组件通常会被多个应用或单个应用中的不同部分共享。只保留一份框架代码库（code base）使得应用易于维护，因为当需求改变时，你只须改变一次。应用框架也可能包含多个层（layer），每一层关于应用要支持的业务都有某些假设（assumption）。其中，最底层包含了没有任何业务假设的框架组件，它们是框架中最通用的组件；层次越往上，其组件依赖的业务假设（business assumption）越多，因此也对业务需求和业务规则的变化越敏感。每当变化发生时，只有那些业务假设被打破的层中的组件需要被修改和测试。因此，让框架的不同层包含不同级别的业务知识，

能够减少因改变业务需求和业务规则所带来的连锁反应。这也使得维护成本降低，因为只须维护因业务规则改变所影响到的代码。更多有关框架分层的知识，我们将在第2章讨论。

没有免费的午餐，应用框架提供了上述好处，但它的开发过程需要额外的代价。

## 1.4 应用框架经济学

你已经了解了框架在支持应用开发方面的出色表现，但尽管应用框架在很多情况下都是最佳之选，这并不意味着选择应用框架总是正确的。在探索应用框架潜力的同时，我们一定也不要忽视了开发应用系统的目的。关于开发和使用应用框架，我们需要考虑两件事情以判断应用框架是否有助于我们达到目的，那就是框架开发和用户培训。

### 1.4.1 框架开发

开发应用框架并非易事，也不廉价。为了开发出高度可用和可扩展的框架，你需要首先找到合适的人选——他们不但是业务领域的专家，还是软件设计和开发的专家。上述两项技能缺一不可：没有业务知识，就无法开发特定业务领域框架层（business-domain-specific framework layer），应用开发人员要依靠该层弥补他们业务领域知识的欠缺；没有软件开发技术知识，就无法将框架思想贯彻到具体的框架代码中去，会导致应用开发人员无法重用和扩展这些代码。开发高质量框架，找到业务领域和软件开发的双料专家是第一关。

无疑，框架的设计和实现需要大量人力资源。开发应用框架要求的技能不同于开发应用。框架的设计者必须决定：应用开发者如何从框架提供的服务和架构中获益；如何去抽象应用中特定的通用逻辑，以使开发者通过框架重用这些逻辑；如何在框架的恰当处提供扩展点（hot spot），以使开发者能插入代码获得特定结果。开发框架的一些工作是很抽象的，并且严重依赖于一个前提：应用开发者将如何使用框架构建应用。一开始

就把所有问题都搞清楚是很困难的，因为在设计时只能猜测：最终应用是什么样子，又是如何被构建以解决业务问题的？有的框架组件，可能最初被认为将被应用的多个部分共享，但最终发现并非如此。有的代码段，可能最初被认为是某场景（scenario）独有的，确实应该在应用层实现它，但最终发现它包含通用的抽象，应当被加入到框架层，以便被整个应用所共享。在已经开始使用框架之后，新的业务需求或变更了的需求可能导致两种情况：新的组件加入框架，或者现存框架组件改变。你可能想到了，为了使框架适应将在其上开发的应用的要求，框架开发过程要经过一系列迭代周期（iteration）。框架的开发生命周期和常规应用的相似，即反复地进行分析、设计、实现和稳定工作。框架开发是非常典型的进化型工作，要求持续开发和相关支持工作，以保证框架的实用性。

### 1.4.2 用户培训

因为框架的用户是构建应用的开发者，所以其用户培训就是训练开发者使用应用框架。为了利用框架，开发者必须有足够的应用框架知识，以及如何使用框架进行开发的知识。但是，学习框架是一个耗时的过程。有几个因素导致了学习应用框架的困难。

应用框架天生就是不完整的应用，其中缺少的部分需要开发者编写应用层代码来补充。然而，在应用开发完成以前，对许多开发者来说，框架本身可能不太容易理解，因为框架的每一部分与其他部分的交互关系，并非自始至终都很明显。

框架中也包含许多控制整个框架处理流程所必需的总控代码。尽管框架的目的之一就是把这些复杂的控制逻辑对开发者隐藏起来，但是在培训时，开发者还是应当理解框架是如何工作的。因为框架中的很多控制逻辑以及类之间的依赖关系，并非直截了当，而且比较复杂，所以理解框架的工作原理并非易事。

应用框架提供了新的编程模型，它包含了很多开发者陌生的 API、服务和配置设置项。开发者需要花些功夫学习应用框架，才能流畅使用它，最终高效地在其上开发应用。

尽管学习曲线是要经历的，但我们可以完善文档，以及演示如何在各种场景使用框架的范例，来加快开发者掌握开发应用框架的速度。对于学习编程来说，一个范例，抵得上一大堆文字和图片。

通过考虑潜在的开发成本和工作量，我们能够权衡出是否需要应用框架。并非所有应用都需要在应用框架之上构建，很多成功的应用并未使用应用框架。有这种情况：你希望投入有限的费用并快速开发出解决方案，这时可能使用框架节约的成本还没有开发框架投入的成本多。相反，也有这种情况：应用框架被多个应用共享，从而总的开发成本显著减少。还有这种情况：你希望今天在应用框架上投资，以为将来的开发提供一个可扩展和可重用的基础平台。总之，就是看应用框架是否有助于你达到项目的预期目标。开发应用框架就像在股市投资；好的投资应该对你的投资目标有利，而不是看它们今天是否赚钱。

## 1.5 小结

在应用框架介绍这一章中，我们了解了应用框架的定义和应用框架的简要历史。之后，我们讨论了如何在应用开发中有效地使用框架，以减小总的开发工作量。然后，我们讨论了以应用框架作为起点来开发业务应用的好处：模块化、可重用性、可扩展性、简单性和可维护性。我们还讨论了开发和使用框架的代价，例如额外的开发工作量和开发者培训问题。知道了开发应用框架的好处和代价，有助于我们做出正确的选择——在具体情况下是否应当使用框架。在下一章，我们将讨论框架的内部组成，以及如何使用面向对象技术来开发既可重用又可扩展的应用框架。