维普资讯 http://www.cqvip.com



# 一种 Delegate 机制的设计与分析*)

崔 琳 许满武 杨 群

(南京大学软件新技术国家重点实验室 计算机科学与技术系 南京210093)

**摘 要** 近年来,面向对象成为软件设计和开发的主流技术,然而基于继承的类结构的面向对象语言越来越显示出难以满足系统中动态应用的需求。本文就我们在实际工作中所遇到的问题,在分析基于继承和设计模式两种实现方法的基础上,提出一种基于 delegate 机制的实现方法。分析和模拟实验结果表明,基于 delegate 机制的实现方法明显优于前两者,可以有效地应用于软件开发。

**关键词** 面向对象,Delegate 机制,继承,设计模式

## Design and Analysis on a Kind of Delegate Mechanism

CUI Lin XU Man-Wu YANG Qun

(State Key Lab for Novel Software, Department of Computer Science and Technology, Nanjing University, Nanjing 210093)

**Abstract** In the recent years, object-oriented techniques have become mainstream in software development; however, traditional inheritance-based languages cannot satisfy the dynamic requirement in large application software systems. For that reason, in this paper we introduce a kind of delegate-based implementation method founded on both inheritance and design-pattern. Analysis and experimental results show that the performance of delegate-based implementation method is superior to others.

**Keywords** Object-oriented, Delegate mechanism, Inheritance, Design pattern

面向对象技术自产生至今已获得了迅猛的发展。但是随着研究与应用的深入,也暴露出一定的问题,其中之一就是传统的基于类继承的面向对象编程语言无法满足大型系统中动态应用的需求,灵活的 delegate 语言机制因而成为面向对象技术的研究的重要课题。

近几年,将简洁和灵活的 delegate 融入到基于类继承的语言中去成为面向对象语言发展的新趋势。例如微软公司开发的 Visual J++和 C# 语言系统,以及德国波恩大学开发的 Lava 语言编译器,都从不同角度支持了 delegate 机制。目前,是否在基于类继承的面向对象系统中引入 delegate 机制仍然是讨论的热点之一。

本文以实际软件系统设计问题为例,在基于继承和基于设计模式这两种较为常用的实现方法的基础上,提出了一种 delegate 机制的实现方法。分析和模拟实验结果表明本文提出的基于 delegate 机制的实现方法明显优于前两者。

## 1 Delegate 的起源与发展

Delegate 的思想最初来源于人们对面向对象语言中类的抽象与规整和原型对象的具体与灵活的认识。1986年,Henry Lieberman 发表论文首先具体论述了 delegate 的概念[1],当时这种概念的提出主要是针对基于原型的面向对象语言的设计,如:NewtonScript、Cecil、SELF 等。后来,delegate 的概念被引入到设计模式,被看作是一种具有与继承同样复用能力的组合方法。由于设计模式依据基于类继承的语言,因此它只是采取引用或指针的方式对 delegate 进行模拟,未能真正实现 delegate 语言机制。目前,微软公司在.Net 开发环境中设计的 C# 语言中又给出了 delegate 的另一种概念。

近20年来,delegate 机制一直是对象式程序设计的研究热点。针对不同的应用,人们提出了若干不同的概念,为避免混淆,在表1中对重要的几种进行简要分析比较。

表1

| | 应用领域 | 概念描述 | 实现技术 |
|---|---|---|---|
| 1 | 基于原型的面向对象语言系统(Henry Lieberman) | 某一对象接收到一条消息时,如果该对象的自身特征与处理该消息并不直接相关,则将消息转递给与之相关的原型对象处理,这个转递消息的过程被称为 delegate。 | 基于原型的面向对象语言摒弃了作为对象抽象的类的概念,代之以具体的对象,功能上是用动态的基于对象的 delegate 代替静态的基于类的继承。 |
| 2 | 设计模式 | delegate 被看作是一种具有与继承同样复用能力的组合方法。 | 设计模式主要用于基于类继承的面向对象语言,所以它只是利用引用或指针的方式对 delegate 进行模拟,并不是真正实现 delegate 机制。 |
| 3 | 微软公司最新发布的 C# 语言 | delegate 实际上是一个能够持有对某个类的共享方法或某个对象的实例方法的引用的类。 | C# 语言中定义的 delegate 是一个系统提供和使用的类,这种类定义一种数据类型,内部包含对静态方法的引用,或一个对象方法的引用。所以,在微软给出的这个概念中,delegate 实际上是对某个被调用方法和某个被操作对象的封装[2]。 |

*)资助项目:国家高技术研究发展计划 No. 2001AA113161;江苏省自然科学基金 No. BK2002080。崔 琳 硕士研究生,主要研究领域为软件方法学与新型程序设计;许满武 教授,博士生导师,主要研究领域为软件方法学与新型程序设计;杨 群 博士研究生,主要研究领域为软件方法学与新型程序设计。

维普资讯 http://www.cqvip.com

本文中涉及的 delegate 不同于设计模式和 C# 语言中所定义的概念，源于1986年 Lieberman 提出来的概念，但特别强调了对消息的接收者 self 的限定[3,4]。定义如下：

1)一个对象，如果含有一个指向另一个对象的指针，则称该对象为子对象(child)，称指针指向的对象为父对象(parent)。

2)如果发送一个消息，消息接收者没有实现响应消息的方法，则消息将自动转递给消息接收者的父对象，这样消息的自动转递过程称为 delegate。消息的自动转递过程中 self 指代的对象应保持不变。

简言之，delegate 就是将 self 始终与消息的接收者绑定的自动转递过程。如图1所示。

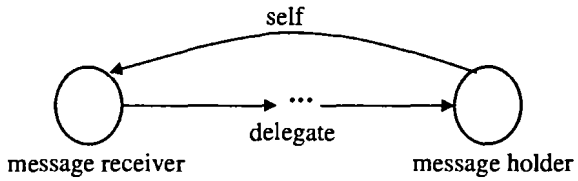


图1

## 2 实现方法

在对 delegate 语言机制研究的过程中，我们发现 delegate 机制具有传统的基于继承的面向对象语言机制所不具备的特性。下面将通过一个 VOD 系统的例子具体说明 delegate 机制的特性及其在实际系统中的实现方法。

VOD(Video-on-Demand，视频点播)系统是一种双向交互式多媒体网络，能够使用户随时随地从网络服务器中访问多媒体数据内容。为了能够提供实时、连续、稳定的音频视频流，同时支持服务器和客户端的 VCR 式的交互，要求系统在网络传输、数据存取以及交互性等多个方面具备提供动态转换策略的能力[5]。利用面向对象技术解决这个问题时，要求动态调用策略对象，delegate 机制在这些方面都有着广泛的应用前景。有关网络传输等方面的内容，我们将在其它文章中另作评述。

本文仅以 VOD 系统中动态转换传输策略的问题为例说明 delegate 语言机制的特性。具体说来，当用户需求量过大导致图像传输率低于给定的阈值时，服务器能够自动转换为速度快但视频质量不高的传输策略，来保证传输速度；当用户需求降低的时候，服务器也应能自动转换到标准的甚至是视频质量更高的传输策略，给用户提供高质量的视频流。

解决这个问题，需要建立几个基本的类：

· VideoServer 类是一个抽象类，它的一个子类 VideoTransfer 负责控制和维护视频服务器的视频流的传输服务。

传输策略并不是由 VideoServer 实现的，而是由抽象的 VideoDisplay 类的子类各自独立的实现的。VideoDisplay 类的各个子类实现不同的传输策略：

·FastDisplay 类实现速度快但视频质量不高的传输策略。

·StandardDisplay 类实现标准的传输策略。

·HighQualityDisplay 类实现视频质量高但速度慢的传输策略。

VideoDisplay 定义了一个方法 display，用来实现传输策略。在它的每个子类中都对这个方法进行了具体定义。

下面将具体给出三种解决问题的方法。

### 2.1 基于继承的实现方法

继承是面向对象系统中类之间常用的一种关系。作为一种模块扩充机制，继承能通过增加或修改已有类的特征去定义新类；作为一种类型精化机制，继承能支持通过实例化已有类型去定义新类型。基于继承的实现方法如图2所示。

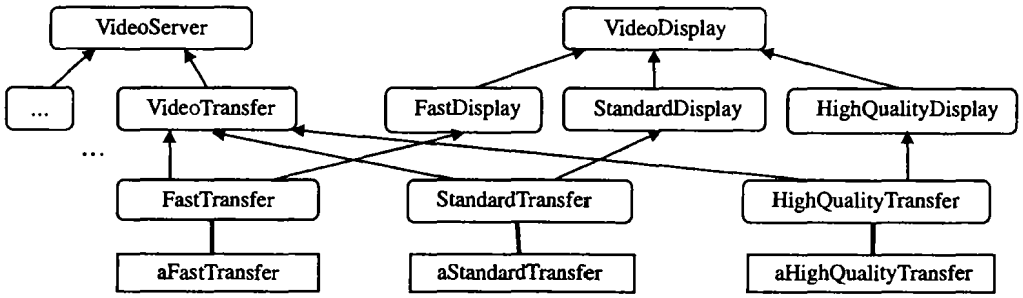


图2

类 FastTransfer、StandardTransfer、HighQualityTransfer 分别是某一种特定的传输策略类的子类，同时也是传输类 VideoTransfer 的子类。在运行时刻，假设系统需要采用速度快的传输策略，则系统生成类 FastTransfer 的对象 aFastTransfer。

### 2.2 基于 Strategy 模式的实现方法

设计模式通过命名、抽象和确定一个通用设计结构的主要方面，使这些设计结构能被用来构造可复用的面向对象设计。设计模式确定了所包含的类和实例，它们的角色、协作方式以及职责分配。设计模式的基本思想是将程序中可能变化的部分与不变的部分分离，尽量减少对象之间的耦合，当某些对象发生变化时，不会导致其他对象都发生变化。

仍然考虑前面讨论的有关传输策略的问题，这次采用 Strategy(策略)模式来实现。Strategy 模式是对象行为型模式的一种，主要通过定义一系列单独封装并可相互替换的算法，使算法可独立于使用它的客户而变化。在 VOD 系统动态转换传输策略的问题上，可以将不同的传输策略看作是不同的算法进行封装。其它类保持一个对它们的父类的引用实现动态的调用。如图3所示。

VideoServer 类维护对 VideoDisplay 对象的一个引用 Videodisplay。一旦 VideoServer 类获得新的运行参数，改变传输策略时，它就将这个职责转递给它的 VideoDisplay 对象。在参照该模式实际编程使用时，这个引用可以由对象引用或者是函数指针来实现。在后面的详细分析中可以看到，这从某种

维普资讯 http://www.cqvip.com

程度上可以看作是对 delegate 的一种模拟。

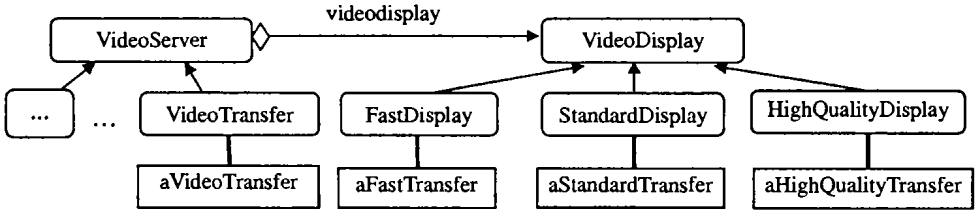


图3

### 2.3 基于 delegate 的实现方法

delegate 的特点就在于它的动态灵活性，这里提出一种基于 delegate 的实现途径，以显示它的这种特性，如图4所示。

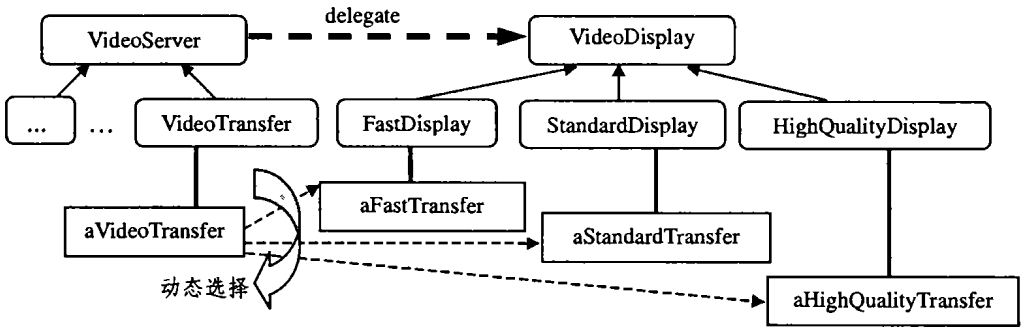


图4

在这个方案中，只需要一个简单的消息 self.display()，通过 VideoServer 类，就能将对方法 display()的调用委派给它的目标对象，也就是 VideoDisplay 的某一个具体子类的对象。VideoServer 类只需要针对不同的情况提供选择的标准，而对不同的传输策略中各不相同的 display()方法的调用则由 VideoDisplay 类完成。假设系统在运行时，aVideoTransfer 对象改变了它的属性，它所要做的只是转递消息 self.display()，其余的都由 delegate 机制和动态绑定完成。

## 3 理论分析对比

本节将对上述提出的基于 delegate 的方法和其他两种实现方法进行分析对比。

### 3.1 基于 delegte 的实现方法和基于继承的实现方法的比较

主要从以下三个方面比较：

·多重继承 在基于继承的实现方法中，FastTransfer 类需要同时获得 VideoTransfer 类和 FastDisplay 类中的信息，这就产生了多重继承。多重继承一直是面向对象语言系统中一个比较棘手的问题，如果不能有效地解决，会存在重复继承的隐患。所以许多面向对象语言并不支持多重继承，例如 C++等。Java 语言系统中也是通过接口间接地实现多重继承。

利用 delegate 解决可能出现的多重继承问题时，可以将概念中不变的和互斥的部分由继承机制建模，而变化的和涉及具体对象共享的部分则由 delegate 建模。这种将继承机制和 delegate 机制相结合的策略避免了多重继承的情况，同时高效地解决了问题。在基于 delegate 的实现方法中，抽象类 VideoDisplay 和各个封装了传输策略的类保持继承关系，而控制系统变化的 VideoServer 类通过 delegate 机制和 VideoDisplay 类保持消息转递关系，从而避免了多重继承。

·动态性 在基于继承的实现方法中，各个类之间的关系是在编译时刻静态定义的，因而无法在运行时刻改变从父类继承的实现。同时，子类和父类间紧密的依赖关系，以至于父类实现中的任何变化必然会导致子类发生变化。而很多情况下，许多操作行为在实例对象生成之前还无法确定。正如在基于继承的实现方法中，在编译时刻就确定下类 FastTransfer、StandardTransfer、HighQualityTransfer 的实现，假设在运行时刻用户希望借用上述几个类中的一些方法产生新的传输策略，这种方法就很难满足这种需求。任何对继承关系的改变，例如增加一个新的实例变量，或者改变类间的结构，都会导致原有数据结构的改变，甚至是整个系统的重新编译。

而 delegate 是通过对其他对象的消息转递机制在运行时刻动态定义的。对象间的关系是通过接口实现的，所以对象间存在较少的依赖关系。直到接收到消息，对象的行为才能被确定下来。所以，动态灵活性被认为是它最大的优点。

·效率 从图2和图4中可以清楚地看到，前者涉及到9个类，而后者只涉及到6个类。从空间效率上来看，基于 delegate 的实现方法比基于继承的实现方法有着很大的优势。让我们再从它们的实现机制上来看，对于继承机制，由于子类要复制父类中的所有特征，因此随着继承层次的加深，对象实例也变得越来越大。而对于 delegate 机制来说，每个对象只是声明和其他对象不同的特征，因此对象的大小并不依赖于类间关系的层次。

### 3.2 基于 delegte 的实现方法和基于 Strategy 模式的实现方法的比较

虽然设计模式的产生给传统的基于类继承的面向对象技术提供了一种利用复用模式进行软件开发设计的有效途径，但是它归根到底只是一种实现模式，并不能从语言机制上支持面向对象程序设计。比较图3和图4，可以看到它们的类的个数大致相同，整体结构也相似，可是在实现机制上却是大相径庭。基于 Strategy 模式的实现方法中类 VideoServer 和 VideoDisplay 之间维护一个引用 Videodisplay，在实际编程时由对象引用或是函数指针具体实现，实际上只是对 delegate 机制的一种模拟。这种实现方法通过一系列相互作用的类表现一种消息传递机制，类间的依赖关系并不是建立在应用程

维普资讯 http://www.cqvip.com

序的语义上，而是仅仅基于模式的技术细节，这种技术上的模拟有可能产生效率低下或不安全的后果。例如，函数指针就不是一种类型安全的机制，而对象引用同继承类似，也存在空间效率低下和不灵活的问题。所以，设计模式的这种模拟并不能代替 delegate。同时，程序设计中增加的依赖和假设关系也经常需要对类的结构和数据关系进行修改，因此固定的设计模式阻碍了软件的复用，也使维护程序的一致性复杂化。

这些问题并不是设计模式本身带来的，而是归因于底层的对象模型并没有提供合适的机制来支持设计模式。例如在基于 Strategy 模式的实现方法中，没有 delegate 机制的支持，类 VideoServer 和 VideoDisplay 之间的接口必须使得定义具体传输策略的子类能够有效地访问它所需要的 VideoServer 类中的数据，反之亦然。一种办法是让 VideoServer 类将数据放在参数中传递给 VideoServer 类的具体子类，这使得类 VideoServer 和 VideoDisplay 解耦，而且还会传递一些冗余数据。另一种办法是让类 VideoServer 将自身作为一个参数传递给类 VideoDisplay，后者再显式的向前者请求数据，或者，后者可以保存对前者的一个引用，而这实际上就是对 delegate 的模拟。

基于 delegate 的实现方法从语言机制上支持 delegate，相比较于基于 Strategy 模式的实现方法，前者明显地改进了基于类的面向对象语言的灵活性，从底层支持具体对象的建模，并可以在运行期间动态改变操作。

### 3.3 将 delegate 和继承以及设计模式的技术结合起来

经过上面的分析，基于 delegate 的实现方法表现出了极大的优势。但是，并不是说要舍弃类继承机制和设计模式的方法。仔细观察和分析图4可以看出，基于 delegate 的实现方法实际上也是遵循着 Strategy 的模式，只不过利用 delegate 机制实现了类间的消息传递关系。同时，这个模型中的抽象类 VideoDisplay 和具体的传输策略的类之间依然保持着继承关系。所以，在使用 delegate 技术时恰当的应用设计模式和继承机制，可以在提供动态灵活性的同时，增强程序的复用性和可理解性，并提高运行效率。下面将简要谈一下 delegate 与这些技术相结合的思路：

·利用设计模式的思想进行面向对象系统的设计，只是将类间存在的需要动态实现的消息传递机制由 delegate 实现；

·将概念中不变的和互斥的部分由继承机制建模，而变化的和涉及具体对象共享的部分则由 delegate 建模。

仍然以 VOD 视频点播系统中的动态传输问题为例，图5简要描述了它们之间的关系：

具体应用：

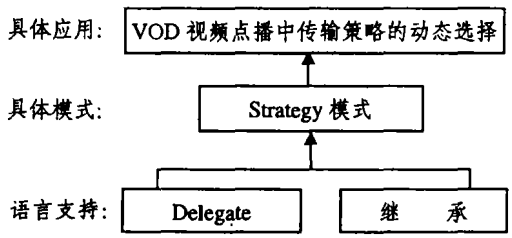


图5

## 4 模拟实验结果

为了具体的比较一下基于 delegate 的解决方法和本文中提到的其它两种实现方法，下面使用这三种方法在 VOD 系统的客户端实现一个简单的动态选择媒体播放器的功能，具体使用的语言环境如下：

·基于 delegate 的实现方法使用德国波恩大学开发的 Lava 语言编译器；

·基于继承和基于设计模式的实现方法使用 Sun 公司开发的 JDK1.4.0开发环境。

在每一种情况下，客户端根据从网络上接收到的媒体格式信息，动态调用本地媒体播放器实现实时播放。选择这个简单的例子的原因主要是因为它的大小——它小到可以很容易理解，大到可以推广到一个更大的具有实际意义的例子。实验结果如表2所示。

表2

| 实现方法种类 | 类个数 | 源文件大小(字节) |
|---|---|---|
| 基于继承的实现方法 | 12 | 3,835 |
| 基于设计模式的实现方法 | 6 | 3,381 |
| 基于 delegate 的实现方法 | 6 | 2,768 |

从表2中可以明显的比较出：

·基于继承的实现方法有12个类，基于 delegate 的实现方法只有6个类。产生这种差别的原因是前者存在多重继承，只有利用 Java 中的接口(interface)来间接实现。

·基于设计模式的实现方法比基于 delegate 的实现方法多产生了20%的源代码，因为前者只是对后者的一种模拟，必然会产生冗余代码。

这个模拟实验不仅可以明显地看出基于 delegate 的实现方法的简洁性，而且也说明了 delegate 语言机制具有动态灵活的特性，因为语言的简洁性是以底层机制的支持为基础的。

**结束语** 传统面向对象系统中的继承机制在编译时刻静态定义类间关系，所以无法有效地动态调用对象，同时还可能产生重复继承的错误，理解性差，空间效率低。采用设计模式模拟实现 delegate 机制弥补了继承机制的部分不足，对引起的代码冗余和不安全等问题可通过一定途径消除。

本文主要从以下两个方面介绍和分析了一种 delegate 机制：

·提出了一种基于 delegate 机制的实现方法，有效地解决了实际应用中动态性问题。

·利用理论分析和模拟试验同基于继承和基于设计模式这两种较为常用的方法进行比较，以说明将这种 delegate 加入基于类继承的面向对象语言系统的必要性和可行性。

Delegate 是面向对象技术与问题求解的认识论紧密相关的一种机制，在实用系统研制中具有广泛的应用前景，我们正在深化相关的工作。

## 参 考 文 献

1 Lieberman H. Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems. 1986

2 Richter J. An Introduction to Delegates. MSDN Magazine, April 2001

3 Kniesel G. Implementation of Dynamic Delegation in Strongly Typed Inheritance-Based Systems, 1994

4 Kniesel G. Dynamic Object-Based Inheritance with Subtyping. 2000

5 Ma Huadong, Shin G. Multicast Video-on-Demand Services. ACM SIGCOMM Communication Review, Jan. 2002