第4章 简单例子概述

在阅读关于CORBA的技术特性时，很难把它们和现实世界问题关联起来。本章引入一个例子，它演示了一些业务需求，这些需求可驱动用户开发一 CORBA系统，并说明如何使用CORBA组件来完成这些需求。在本书中还会用这个例子来阐明 CORBA设计和开发的特殊方面。这里引入的课题在以后的章节中会有更详细的探讨。

4.1 简介

Goldman Sachs、Lehman Brothers和Merrill Lynch这些投资银行每天转手数以十亿计的美元，为使业务成功，他们需要密切注意世界经济的趋势。在早餐时阅读华尔街报是每个投资经理的职责。但交易者从哪里可获取过去和现在的金融事件统计信息呢？他们需要哪些信息来确认自己的投资战略呢？ Bloomberg、Reuters和Quotetron这些公司处理金融信息，把这些重要的财富售给全世界所有的 J.P.Morgans和Merril Lynches。但如果这些金融信息供应者把相同的信息售给所有的投资银行，那么这些银行如何从中区分自己的业务呢？现在大多数的投资银行严重依赖于他们自己的 IT部门所提供的服务，这些服务提供的系统可被经纪和交易者使用，以使自己在业务中成功。使金融信息能以多种不同的方式被获取是这个竞赛的重要部分。

金融信息供应者提供的第一个系统经常和前端捆绑在一起。例如，信息供应者不仅提供一物理连接来访问信息，还有提供了 stock ticker服务的终端，或类似的一些东西。显然，投资银行感兴趣的是把信息输入到自己的系统，以获得使业务成功所需的竞争信息。使银行的IT部门能直接访问信息服务的第一个系统经常只是简单地提供一物理连接，如出租的线路，它可以通过低级的网络协议如 TCP/IP来访问服务。金融信息供应者还提供消息和数据格式的规范，用于和信息服务的交互。这些规范是长而复杂的文档，例如使用类似于 BNF语法的东西来描述不同的格式。每次信息服务功能改变时，规范格式也要改变，而且用户必须改变他们系统的实现来适应新格式。由于这种“未加工套接字”方法的限制，一些信息供应者开始提供给他们的顾客程序库，并在其中封装了网络消息的格式，这使得访问信息服务变得更容易。尽管这是重要的进步，但它并没有真正解决问题。库使用供应商独有的 API，它们难于使用而且必须为新的编程语言和编译器版本以及操作系统进行改造。

图4-1在很高的层次上显示了金融信息用户和信息供应者的交互。这些用户可能包括多种类型的组织，例如交易者、投资经理、金融分析员或新闻机构，每一种都以不同的方式来使用信息。一条组织上的分界线存在于信息供应者和信息最终用户之间。因此，信息供应者预先不知道用户如何使用它的服务。这意味着信息供应者必须定义足够灵活的接口，以满足所有不同种类的用户要求。传统的系统中，在这种灵活性和开发客户机应用程序的困难性之间要有一个取舍。金融信息供应者通常提供历史数据，通过使用拉（ pull）类型通信来访问，而后续信息的更新则通过推（ push）类型通信来提供。如图 4-1所示，客户机拉历史数据[1]，而服务器推股票价格的更新[2]。

使用基于CORBA的方法，供应者可以用IDL定义它的信息接口。供应者然后向用户给出IDL定义和访问服务器的权利。用户可以采用CORBA标准的优点和面向对象编程方法来创建符合需求的客户机应用程序。这些应用程序可使用标准的TCP/IP连接在拨号线路上，或是在Internet上来访问金融信息。

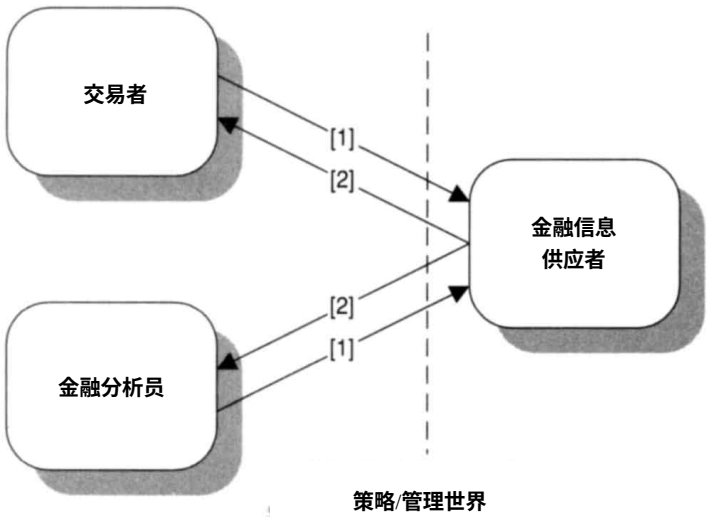

图4-1 金融信息供应者

通过CORBA IDL，由金融信息供应者提供的输出服务有几个主要的好处。CORBA IDL是可接受的定义接口的标准，这样程序员就不必为每个新系统而学习指定系统的数据、消息格式和访问机制；CORBA利用了企业级面向对象的优点，这意味着程序员可着重于创建功能丰富的应用程序，而不用担心网络级的编程。CORBA IDL编译器自动生成使接口可被客户机访问所需要的代码，以及在系统中集成服务器端实现所需的代码。CORBA IDL编译器对大量的编程语言和操作系统可用。所有这些都意味着服务供应者不必维护、移植(port)和分配专有的类库，但可依靠CORBA作为定义和访问远程服务的标准方式。

本章的余下部分介绍了一个在金融信息服务和股票交易领域中的例子。这个例子包括两个组件。第一个组件通过提供关于股票价格的信息来供应简单的金融信息服务。第二个组件是证券管理系统，它使用金融信息服务来提供像计算特定投资的当前价值等的服务。第一个组件可能由金融信息供应者提供，而第二个组件则由投资银行为内部的使用而实现，或甚至是由零售经纪实现，并可能会在Internet上提供证券管理。

4.2 StockWatch组件

第一个演示组件称为StockWatch例子，最初只提供对历史股票数据的访问。服务器假设客户机想选择一特定的股票，然后再访问该股票价格的历史。虽然StockWatch服务可被任意种类的CORBA客户机使用，但金融信息供应者只为演示目的而提供一简单的客户机实现。演示客户机实现简单的GUI，它显示可选择股票的代号，并根据所选择的特定代号来显示股票的价格历史。图4-2是简单的GUI客户机的概貌。

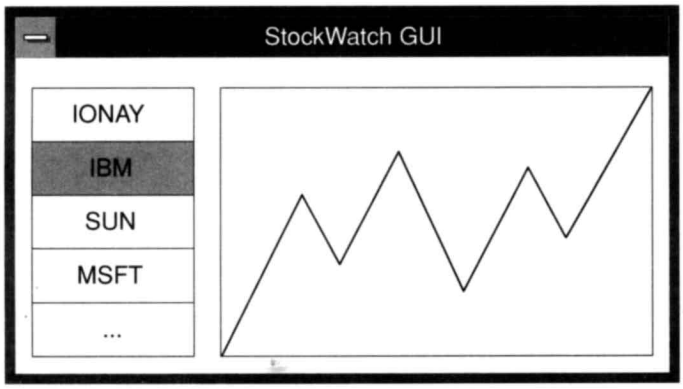

图4-2 StockWatch GUI

GUI客户机和金融信息服务器通信，这个服务器依次访问数据库来获取股票信息。服务器实现两类CORBA对象：StockWatch对象，它允许客户机选择特定的股票；Stock对象，它提供对特定股票信息的访问。在使用GUI客户机时，最初的系统有一简单的三层体系结构，如图4-3所示。

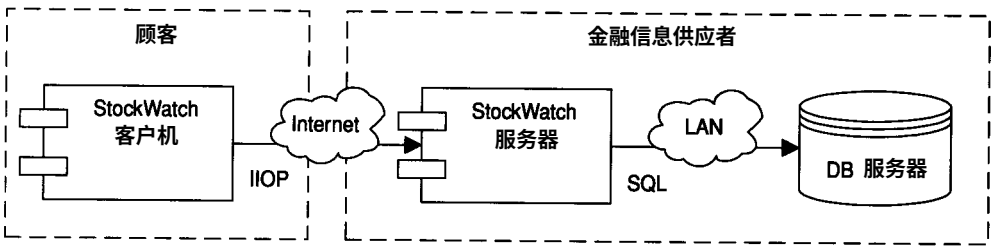

图4-3 StockWatch概貌

4.2.1 StockWatch接口

StockWatch和Stock对象的IDL如下所示：

```
// IDL : StockWatch Example

//
// Basic type definitions
//
typedef string Date;
typedef float Money;
typedef string Symbol;
typedef sequence<Symbol> SymbolSeq;


//
// Exceptions
//
exception rejected {
    string m_reason;
};
```

```
//
// Structs
//
struct PriceInfo {
    Money m_price;
    Date m_when;
};
typedef sequence<PriceInfo> PriceInfoSeq;


//
// Interfaces
//
interface Stock {
    Symbol getSymbol();
    string getDescription () raises (rejected);
    Money getCurrentPrice () raises (rejected);
    PriceInfoSeq getRecentPrices () raises (rejected);
    };

interface StockWatch {
    SymbolSeq getSymbols () raises (rejected);
    Stock getStockBySymbol (in Symbol aSymbol)
      raises (rejected);
};
```

客户机通过获取StockWatch对象的引用来首次接触金融信息服务器。StockWatch接口包括两个操作：

- getSymbols()返回所有已知股票代号的列表。
- getStockBySymbol()返回和特定代号相关联的stock对象。

StockWatch对象作为金融信息服务的进入点，允许客户机获取与特定股票相关联的Stock对象的引用。

当客户机得到Stock对象的引用后，它就可以查询对象来取得相应股票的信息：

- getCurrentPrice()返回股票的当前价格。
- getRecentPrices()返回近来股票价格的历史。
- getDescription()返回股票的文字描述。

4.1.2 数据库模式

为获取股票价格的信息，服务器使用嵌入的SQL和关系数据库管理系统交互。下面显示了用来存储股票和股票价格的数据库表：

STOCK 表	
string SYMBOL	string DESCRIPTION

STOCK_PRICE 表		
string SYMBOL	number PRICE	date DATE

STOCK表使用股票代号作为主键。STOCK_PRICE表使用股票代号和日期的组合。在STOCK_PRICE表中，SYMBOL列也是一外部关键字，它使股票价格和STOCK表中的输

入相关。

这样的数据库中数据量可能是很大的，这是由于股票数量很多，而且还必须存储历史数据。另外，表的内容可能很频繁地改变，股票的价格每天都要改变多次，而且这些改变必须在长时间内捕捉。

4.2.3 扩展StockWatch的系统体系结构

显然，类似这里所描述的金融信息服务器必须能够处理大量的数据和网络通信。一个现实世界的系统可能有大量的客户，要求高的性能，甚至是实时的信息更新。在这些限制下，存储和传递如此大量的数据就会极其困难。但在本节中会描述 CORBA如何使应用程序规模达到现实世界的级别。

1. 使用负载平衡增加可伸缩性

我们的金融信息服务可能被大量的客户访问。如果我们想实现应用程序真正的可伸缩性，就不能依赖单独的服务器进程来同时向所有客户发送信息。为保持可接受的性能级别，我们要把处理负载平衡到一些服务器进程中。

在这个例子研究中，要使用一个基于命名服务的负载平衡机制，对此在第 15章“负载平衡”中有更详细的描述。基本的思想是负载平衡使命名服务允许多个对象以同一个名字注册，这样单独的一个名字就代表一组对象，这些对象提供相同的接口和功能。如果客户解析一个名字，命名服务就要选择其中一个对象，例如使用简单的随机选择策略。这个简单的负载平衡机制称为对象组模式。

图4-4显示了StockWatch系统中基于对象组的负载平衡。每个客户机连接到金融信息服务供应者的命名服务。客户机然后激发根命名上下文上的 resolve()操作，传递它想要接触的StockWatch对象的名字。负载平衡使命名服务可返回相应对象组中某个对象的引用。在本例中，客户机在其整个生命周期间使用一单独的服务器进程。用户很容易想象其他情况 —— 例如，客户机中的哪一个从服务器进程池中获取 Stock对象。在第15章“负载平衡”中会对此进行更深入的探讨。

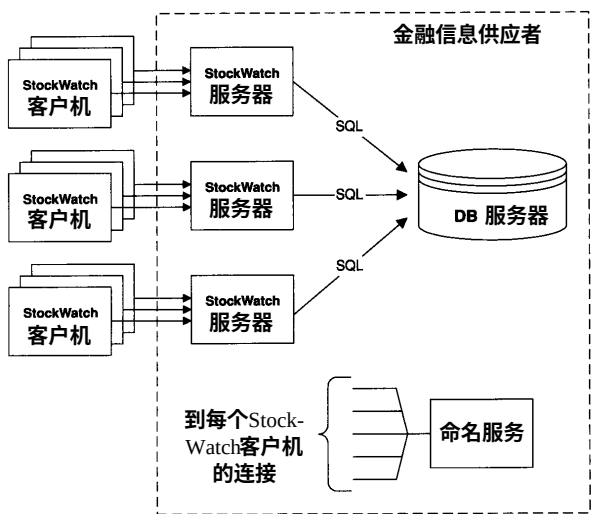

图4-4 StockWatch的负载平衡版本

2. 使用事件通知公布股票价格的改变

在StockWatch例子中，考虑一下这种情况：金融信息供应者想向客户提供实时的股票价格变动。服务器必须能够在和用户应用程序没有紧密耦合的情况下向任意数量的用户发送变动信息。要实现这个目标，就要使用CORBA事件服务。

事件服务允许用户使用事件信道来去掉客户机和服务器之间的通信，这在第 7章“消息接发”中描述。在这个例子中，每只股票有一个事件信道是合理的。用户为所有他们感兴趣的股票订购事件信道。服务器使用合适的事件信道向客户机发送价格变动。这个体系结构如图4-5所示。

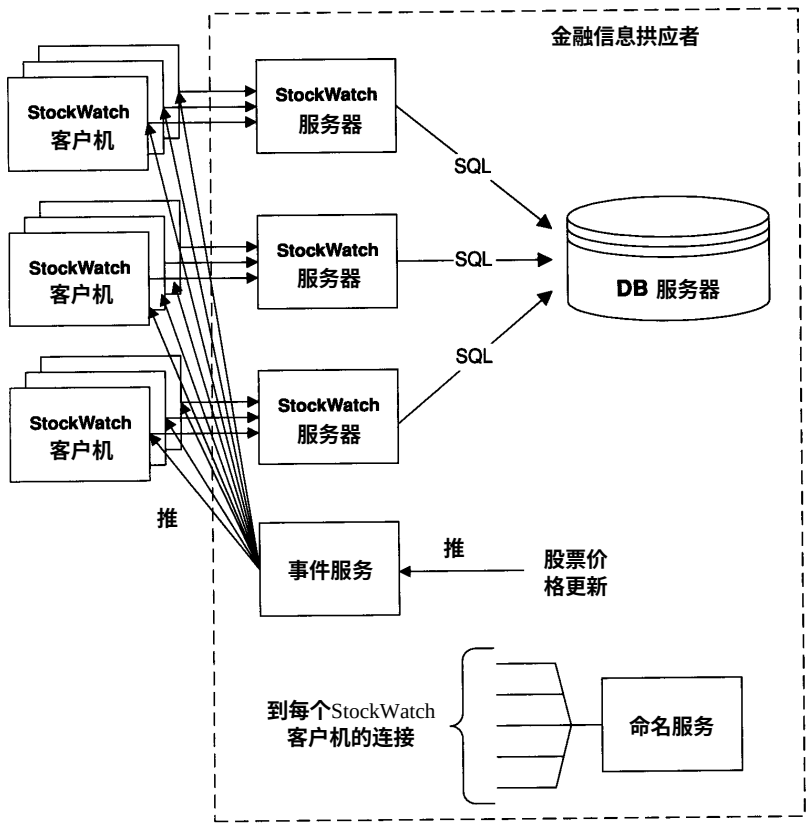

图4-5 具有事件通知的StockWatch

为实现这个体系结构，如前所述，要向Stock接口加入一个操作。这个操作名为getFeed()，它允许客户机获取与Stock对象相关联的事件信道的引用。这个操作的IDL定义如下所示：

```
// IDL
interface Stock {
    // Operations as before.
    // …

    // Operation to obtain the event channel
    // administration object, for event consumers.
    CosEventChannelAdmin::ConsumerAdmin getFeed();
};
```

负责管理股票价格变动的机制会向和特定股票关联的信道反馈每次价格变动。事件服务把价格变动推向所有的用户，这些用户都以特定的信道被注册。

4.3 证券管理器组件

在本书中，在描述CORBA设计和开发问题时，就要参考StockWatch应用程序。为了描述其中的一些问题，必须扩展这个例子以允许对更多高级设计问题的讨论。在这里描述一合适的客户机应用程序，称为证券管理器（Portfolio Manager）组件。

想象一下一间股票经纪公司想向其客户提供基于Internet的证券管理系统。公司向其客户提供一简单的GUI，用于和后台的证券管理器通信。GUI以Java小应用程序实现，并使用Java ORB。证券管理器是一CORBA服务器，它实现一些分布式对象。

图4-6显示了证券管理器的类图。在UML术语中，证券聚集了一系列的控股。一个控股指用户所持有给定股票的股份。通过规定相应股票的代号就能够取得具体的控股。最后，证券提供一个方法来取得证券的当前价值，这是所有控股当前价值的总和。

证券管理器GUI如图4-7所示。这个简单的GUI允许客户机访问应用程序中所有可用的功能。其中有三个标签：查看、买入和卖出。在本节中忽略了买入和卖出功能。查看标签显示了证券的总价值，还允许用户得到关于每个控股的详细信息。当前控股以一股票代号列表表示。用户可从列表中选择一个代号以得到关于特定控股的更多信息。

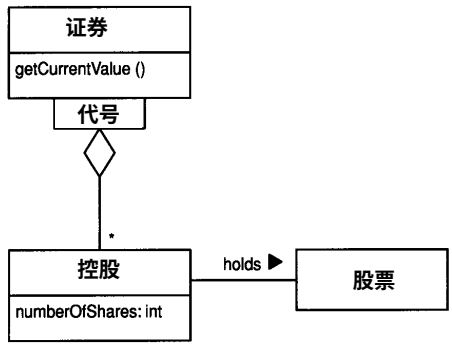

图4-6 证券管理器类图

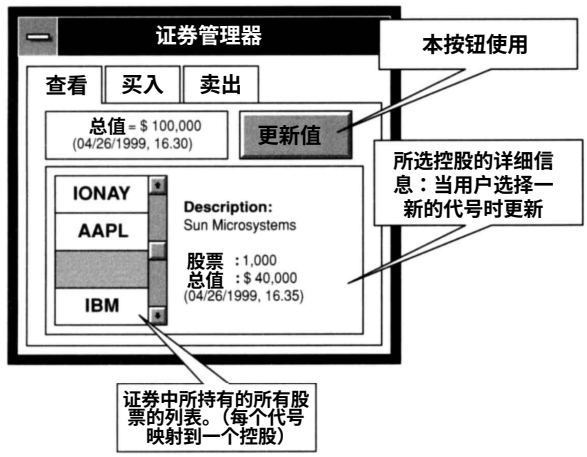

图4-7 证券管理器GUI