第 32 卷 第 19 期 计 算 机 工 程 2006 年 10 月
Vol.32 № 19 Computer Engineering October 2006
·软件技术与数据库· 文章编号：1000—3428(2006)19—0062—03 文献标识码：A 中图分类号：TP311.52

MVC 模式在分布式环境下的应用研究

陈 乐，杨小虎
(浙江大学计算机学院，杭州 310027)

摘 要：随着分布式技术的发展，MVC 的含义和用途变得更加广泛，不仅可以用于组件的构造，也可用于分布式程序的设计。文章介绍了传统的 MVC 模式，分析了在分布式环境下如何合理、有效地应用 MVC 模式，并在此基础上结合 Web Services 和消息中间件技术，提出了一个分布式 MVC 模式的实现框架。该框架可以帮助创建结构良好、松散耦合的分布式应用。

关键词：MVC 模式；分布式应用；表示层模型；域模型；Web 服务；消息中间件

Research of Applying MVC Pattern in Distributed Environment

CHEN Le, YANG Xiaohu
(College of Computer Science, Zhejiang University, Hangzhou 310027)

【Abstract】 As the distributed technique develops, the use of MVC becomes wider and wider, MVC is no longer limited to the construction of components, but also can be used in the software design of distributed application. This paper introduces basic concepts of MVC pattern, analyzes how to apply the MVC pattern properly and efficiently in distributed environment and provides an implementation framework for distributed MVC pattern using Web services and message oriented middleware. This framework can help to build a well-architected, loosely-coupled distributed application.

【Key words】 MVC pattern; Distributed application; Presentation model; Domain model; Web services; Message oriented middleware

1 概述

MVC(Model-View-Controller)是由Smalltalk-80 引入的一种面向对象的设计模式，用于创建可重用的界面程序[1,2]。MVC模式将用户界面程序分为 3 个部分：模型(Model)，视图(View)和控制器(Controller)，三者之间的关系如图 1 所示。

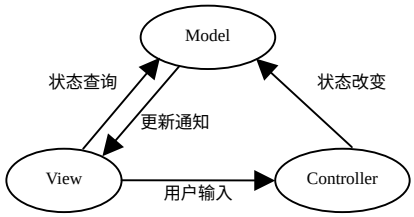

图 1 MVC 模式

模型负责维护现实世界的状态并提供对这些状态进行查询和操作的接口，当状态改变时，它会通知相应的 View 进行更新。视图负责以图形或文字的方式将 Model 的状态显示给用户。控制器解析用户的输入(如鼠标和键盘事件)，然后调用 Model 的相应接口来改变其状态。

MVC 模式将数据表示、输入控制和数据处理分离开来，使得程序结构清晰并提高了各部分的重用性，在传统界面程序设计中得到了广泛应用。

随着网络技术的发展，计算机软件已经进入了分布式应用的时代。MVC 模式对于界面设计中普遍会遇到的基础问题作了很好的解答，其 Model、View 和 Controller 分离的思想对于分布式程序来说依然适用。

2 在分布式环境下应用 MVC 模式

MVC 模式在概念上的定义是很丰富的，但是为了维持模型的一般性，MVC 模式对于许多实现上的问题并没有给出明确的定义，不同的环境下可能有不同的实现方式。下面针对分布式应用的特点，讨论分布式环境下如何合理、有效地应用 MVC 模式。

2.1 Presentation Model 和 Domain Model

当软件开发人员将MVC模式运用到实际项目中的时候，他们发现显示系统的状态和操作在很多情况下和底层的商业逻辑的接口并不一致，将其封装在另一个更接近View的“Model”中可能更加合适。这样在系统中就存在两个Model，一个是和显示相关的，称之为Presentation Model(或者Application Model)[3]；另一个是和商业逻辑相关的，称之为Domain Model。图 2 给出了带有Presentation Model的MVC模式。

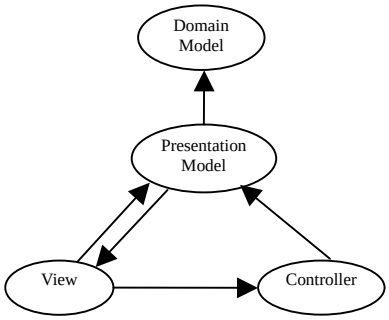

图 2 带 Presentation Model 的 MVC 模式

Presentation Model 的状态和操作是面向显示层的，主要分为两种情况：第 1 种是商业无关的，例如界面的布局、字

作者简介：陈 乐(1981－)，男，硕士，主研方向：软件工程和电子商务；杨小虎，博士、副教授

收稿日期：2005-10-17 **E-mail：**chenle81@yahoo.com.cn

体等，这些状态的改变不会影响到 Domain Model，也不需要和其它 View 同步；第 2 种是商业逻辑相关的，例如记录的插入、删除，当 Presentation Model 接收到这些状态的操作请求后，需要将其转化为 Domain Model 能够理解的信息并调用 Domain Model 的方法来进行逻辑计算，其改变也会通知给相应的 View 以实现同步，在这种情况下，Presentation Model 实际上相当于在 View 和 Domain Model 之间起了一个中介者(Mediator)的作用。

Domain Model 的状态和操作是面向商业逻辑的，用于实现真正的逻辑计算。Domain Model 一般需要进行真正的数据存取操作(例如数据库操作)。

可以看出，Presentation Model 更接近用户，而 Domain Model 更接近商业逻辑，这两部分的分离使得部件的重用性进一步增加，程序的结构更加清晰，并且有利于确定分布式程序各个功能模块的合理分布。

2.2 MVC 功能部件的合理分布

在分布式环境下运用 MVC 模式，一个很重要的工作就是确定各个功能部件如何分布，即 Domain Model、Presentation Model、Controller、View 这几部分如何在客户机和服务器之间划分。下面分析各种不同的划分方式：

(1)Domain Model、Presentation Model、Controller 和 View 都在客户端。将所有的显示逻辑和商业逻辑都放在了客户端，而服务器端只是作为一个原始数据存放中心(例如是一个数据库服务器)，负责简单的数据读写。这种划分下客户端的工作量非常大，和服务器端之间的数据通信比较频繁，从而造成程序效率低下。

(2)Domain Model、Presentation Model 和 Controller 在服务器端而 View 在客户端。客户端只负责界面的显示，对于用户输入的解析和商业逻辑的处理都在服务器端进行。这种划分的一个典型代表就是目前取得广泛应用的 B/S 构架，在 B/S 构架中，客户端主要指网页浏览器，负责显示 HTML 页面；用户的输入都将发送到服务器端进行处理，服务器动态生成新的 HTML 页面作为结果返回给客户端。这种划分没有充分利用客户机越来越强大的计算能力，无法在客户端实现复杂的用户交互体验；实际上对于许多商业逻辑无关的用户操作，完全可以在客户端实现。

(3)在 Presentation Model 层和 Domain Model 层之间划分系统功能，即 Presentation Model、Controller 和 View 在客户端而 Domain Model 在服务器端。在这种划分下，客户端负责处理接近用户的界面逻辑，可以充分利用客户机的计算能力提供给用户丰富的交互体验；另一方面，服务器端负责处理商业逻辑运算并连接数据库服务器进行数据存取。这种划分方式使得整个系统在客户端有良好的响应，并且充分体现了 MVC 模式的结构清晰和重用性高的特点，使得整个构架易于维护，是一种合理的分布式程序功能划分方式。

2.3 用 Web Services 定义 Domain Model 接口

在单机应用程序中，MVC 的各个部件都处在同一操作系统平台之上，Presentation Model 通过方法调用的方式和 Domain Model 进行通信，这种方式依赖于特定的操作系统平台(如 Windows 系统)来实现。然而现在的分布式程序规模越来越来大，单一的平台已经无法满足实际需要；分布式程序往往要求能够跨平台、跨语言进行操作。为了适应分布式的这种需求，需要实现异构平台下对 Domain Model 的访问。

Web Services技术通过基于XML的SOAP协议，在现有的各种异构平台的基础上构筑一个通用的与平台、语言无关的技术层，实现了程序逻辑之间的松耦合，是创建可互操作的分布式应用程序的新平台[4,5]。

Web Services 对于其他异构平台解决方案(如 CORBA 和 DCOM)的一个重要改进就是它是基于开放标准的，是松散耦合的。松耦合系统的好处在于可以灵活地适应不断变化的业务需要；而紧耦合意味着应用程序的不同组件之间的接口与其功能和结构是紧密相连的，因而当需要对部分或所有组件进行某种形式的更改时，整个系统就显得非常脆弱。

将 Domain Model 的接口发布成为 Web Services，使得显示层和逻辑层之间可以实现跨平台、松耦合的连接，从而使得分布式 MVC 模式能够满足分布式应用的需求。

3 分布式 MVC 模式的实现框架

基于上面的分析，笔者提出了一个分布式 MVC 模式的实现框架(见图 3)。

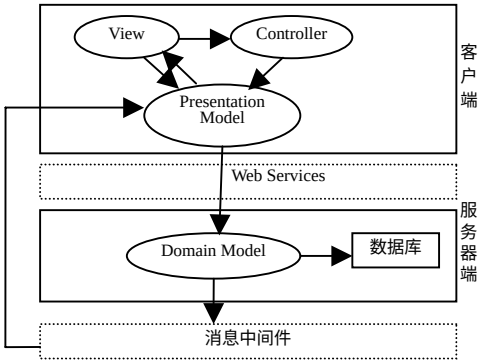

图 3 分布式 MVC 模式的实现框架

该框架在 Presentation Model 和 Domain Model 之间划分系统功能，用 Web Services 封装 Domain Model 接口，并通过消息中间件实现各个客户端的信息同步。框架分为以下几个层次：

(1)客户端

Presentation Model、Controller 和 View 在客户端实现。对于同一个服务器端可以有多个客户端存在，所有客户端都要在服务器端的 Domain Model 上实现同步。

当 Controller 检测到用户的输入动作时，它会调用 Presentation Model 相应的方法。这时可能有两种情况：1)被调用的方法是商业逻辑无关的，则不需要调用 Domain Model 的方法，也就不用发送请求到服务器端，只需要在 Presentation Model 内处理即可；2)被调用的方法是商业逻辑相关的，Presentation Model 需通过 Web Services 调用服务器端 Domain Model 的方法，以完成对用户输入事件的响应。

(2)Web Services 层

Domain Model 将自己的接口封装成 Web Services。Presentation Model 通过访问 Web Services 调用 Domain Model 的方法。Web Services 层为客户端和服务器端提供了跨平台、松耦合的连接。

(3)服务器端

服务器端负责维护 Domain Model 和数据访问。当然数据访问操作也可以独立出来成为数据访问层，这样便形成三层结构。

当一个客户端的操作改变了 Domain Model 的状态之后，Domain Model 需要将自身状态变更的消息发送给相应的客户端，这一消息发送的工作由消息中间件完成。

(4)消息中间件层

消息中间件(Message Oriented Middleware，MOM)是一种特定的中间件，能够提供及时可靠的消息通信，为异构平台下的分布式应用提供了松耦合的信息交换机制[6,7]。

这里主要利用消息中间件的发布/订阅方式来实现Domain Model同步消息的发送。服务器端作为消息发送者，客户端作为消息订阅者，在同一个主题上注册。当Domain Model的状态改变之后，服务器端将状态变更的消息发送到主题上；客户端即可以从主题上接收到该同步协作消息，然后根据消息内容更新自己的显示。

4 应用案例

在一个实时股票交易系统中，笔者运用上述分布式MVC框架进行系统设计，取得了良好的效果。在该系统中，交易者(Trader)可以发送买进或卖出股票的请求；而由交易协调员(Coordinator)根据一定的规则将2个或多个请求进行匹配，以完成股票交易。

该系统基于Java平台实现，其中界面实现基于Swing技术，Web Services采用Apache Axis实现[8]，消息中间件则采用的是Sonic公司的SonicMQ。

4.1 系统设计

整个系统的结构设计如图4。

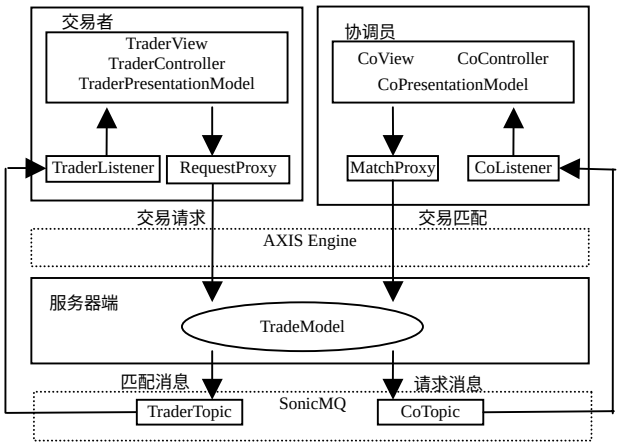

图4 实时股票交易系统结构

交易者的界面可以让交易者查看当前股票的信息并提交股票交易请求，由TraderView，TraderController和Trader PresentationModel实现。协调员的界面可以让协调员查看当前所有交易请求并执行交易匹配，由CoView，CoController和CoPresentationModel实现。

Axis Engine负责将TradeModel发布为Web Services。交易者端通过RequestProxy来调用交易请求的Web Services；协调员端通过MatchProxy来调用交易匹配的Web Services。

在SonicMQ上维护CoTopic和TraderTopic两个主题。服务器端通过CoTopic发送交易请求消息给协调员，通过TraderTopic发送交易匹配成功的消息给交易者。Trader Listener负责接收交易匹配成功的消息并改变Trader PresentationModel的状态；CoListener负责接收交易请求的消息并改变CoPresentationModel的状态。

4.2 系统工作流程

整个系统的工作流程如下：

(1)交易者在TraderView上执行交易请求操作后，由Trader PresentationModel通过RequestProxy调用服务器端的Web Services来提交请求。

(2)Axis Engine接收到交易者的请求之后，向TradeModel中添加1条交易信息，服务器端会发送1条交易请求消息到CoTopic中。

(3)当CoListener从CoTopic上接收到这条消息后，向CoPresentationModel中添加这一请求，这会触发CoView的更新，协调员就可以看到这条交易请求。

(4)协调员在CoView上执行交易匹配操作后，由CoPresentationModel通过MatchProxy调用服务器端交易匹配的Web Services。

(5)Axis Engine接收到交易匹配要求后，调用TradeModel进行匹配操作，接着服务器端会发送一条交易匹配消息到TraderTopic中。

(6)每个相关交易者的TraderListener接收到这条消息后，都会改变TraderPresentationModel的状态，从而交易参与者都可以通过TraderView看到交易完成的信息。一次交易过程结束。

5 结束语

本文在传统MVC模式的基础上，结合分布式应用的特点，提出了分布式MVC框架，该框架具有以下优点：(1)View、Controller、Presentation Model和Domain Model的分离可以帮助程序确立良好的系统功能划分，提高了各部件的重用性；(2)便于在开发组中进行分工，有利于程序的并行开发。通过良好地定义各部件之间的通信方式和接口，可以分配具有不同知识技能的人分别进行不同部件的开发；(3)Web Services和消息中间件技术的使用，实现了松耦合的连接，使整个系统可以轻松应对不断变化的业务需要，有利于系统维护和系统扩展。该框架为分布式应用提供了一种良好的构架解决方案。

参考文献

1 Burbeck S. Applications Programming in Smalltalk-80: How to Use Model-View-Controller (MVC)[EB/OL]. http://st-www.cs. uiuc.edu/users/smarch/st-docs/mvc.html, 1992.

2 任中方, 张　华. MVC模式研究的综述[J]. 计算机应用研究, 2004, 21(10): 1-4.

3 Brown K. Remembrance of Things Past: Layered Architectures for Smalltalk Applications[EB/OL]. http://members.aol.com/kgb1001001/ Articles/LAYERS/appmod.htm, 1995.

4 柴晓路. Web服务架构与开放互操作技术[M]. 北京: 清华大学出版社, 2002.

5 Chappell D, Jewell T. Java Web Services[M]. CA: O'Reilly & Associates, 2002.

6 苏　洋. 分布式应用对象间JMS消息服务原理与消息处理过程[J]. 微型机与应用, 2002, 21(8): 9-l2.

7 SUN Microsystems Inc. Java Message Service Specification, Version 1.1[EB/OL]. http://java.sun.com/product/jms, 2002.

8 Axis Project. Axis Developers Guide[EB/OL]. http://ws.apache.org/axis/ java/developers-guide.html, 2004.