用 PowerBiulder 建立基于 C/S 的分布式应用

尹晓勇　于　洁　(军械工程学院计算机教研室　050003)

摘要:本文从网络应用发展的需要介绍了分布式应用结构,重点论述了 PowerBiulder 的分布式应用技术,提出了商业逻辑分布的基本原则,并结合示例说明用 PB6.0 设计分布式应用的基本方法。

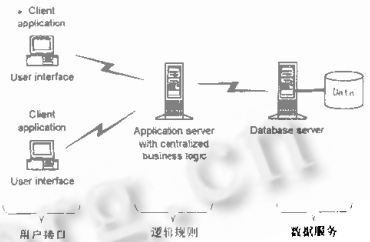
关键词:分布式应用　Client/Server

一、引言

在传统的基于客户机/服务器(Client/Server)体系结构的网络应用中,应用逻辑通常驻留在客户机上并与用户接口交织在一起,使得系统缺乏良好的安全性、可维护性和可扩充性。为了访问数据库,每个客户机必须保持同数据库服务器的连接,当客户机数量增加时,就会导致服务器的负担加重和网络性能下降。解决这一问题的办法是将应用逻辑分离出来,采用对象技术和分布式技术,将应用系统的各个组件按功能和作用配置到网络上的不同机器中,从而充分地利用系统资源和发挥计算机网络的优势,满足大型组织的商业应用需要。

二、分布式应用结构

分布式应用是一种基于 Client/Server 结构的多层应用方案,它采用自然的方式将一个商业应用划分为用户接口、逻辑规则和数据服务三个部分,如下图所示。



用户接口在客户机上负责与用户交互并根据用户请求调用相应的逻辑规则;数据服务由专用的服务器提供数据库管理和操作;逻辑规则驻留在一个或若干个应用服务器上,执行具体的事务逻辑并通过 SQL 等方式向数据服务提出数据或其他资源的请求。在系统实现时将三个部分用相应的功能组件来表示,并分别配置到若干个服务器和客户机上,就形成了以服务器为中心的分布式

应用系统。这样做的优点主要体现在以下几个方面：

(1)维护和管理容易。客户端只处理用户接口，不涉及数据的存取；而商业逻辑的更新只对中间层修改，不用更新整个系统；各层的功能相对独立，既降低了维护和管理的工作量，又易于系统的扩展。

(2)逻辑组件可共享和重用。商业逻辑可用 VC、VB 等工具开发成可重用的二进制组件，供其他应用程序共享和调用，且可镜像到多台服务器上同时运行；另外，应用程序组件还可共享与数据库的连接，降低了数据库服务器的负担，提高了性能。

(3)安全性更高。安全管理基于组件来授权而不是授权给用户，客户机不再直接访问数据库，提高了系统数据的安全性。

(4)开发效率高，有利于团队的分工合作，可灵活地选择不同层、不同组件的开发工具。

三、PB 的分布式应用技术

PB 的分布式应用由两种分别位于不同机器上的应用组成，即：

·服务器应用

·客户机应用

通常，客户机应用处理用户接口而服务器应用为一个或多各客户机提供后台服务。一个服务器应用可以作为另一个服务器应用的客户机，而其本身又是某个客户机的服务器，客户机应用与服务器应用协作，共同完成商业任务。

1.服务器应用

服务器应用包含两个主要组件：

(1)传输对象(Transport object)。传输对象是一个不可视(nonvisual)的对象，它含有处理客户请求连接到服务器应用的参数，并控制服务器与客户机之间的全部通信。

(2)远程对象(Remote objects)。远程对象是不可视的定制类用户对象，它被包含在一个远程服务器上的应用程序中，用于向客户机提供服务。PB 提供同步和异步两种方法供客户机调用远程对象，并且调用可跨越进程边界或计算机的边界，这为分布式计算和多层应用提供了技术。

2.客户机应用

客户机应用包含三个主要组件：

(1)用户接口(User interface)。用户接口定义了所有与用户交互的窗口、菜单和响应用户事件的脚本。

(2)连接对象(Connection object)。连接对象含有建立客户机连接的信息，它标识服务器应用、应用驻留的机器和通信驱动程序，使客户机应用可以连接到服务器应用并请求其服务。连接对象在客户机脚本中实例化。

(3)远程对象类定义(Remote object class definitions)。每一个远程对象在客户机应用中都有一个对应的类定义，这个远程对象的定义的本地副本与服务器中定义的名字相同。在运行时，本地类定义允许客户机象在本地实例化一样地使用远程对象实例。

在客户机上的类定义可以包含远程对象的全部实现，或者包含一个提供远程对象接口表示的代理对象(Proxy object)。对于分布式处理，仅要求客户机上有代理对象。

3.通信驱动

在分布式应用中要视系统工作平台来选择相应的通信驱动程序，PB6.0 提供了以下三种通信驱动程序：

(1)WinSock。通过 Windows 套接字工具提供在 TCP/IP 网络上的通信服务。

(2)NamedPipes。通过命名管道工具提供通信服务。

(3)Local。允许分布计算以本地方式运行。

四、分布式应用的设计

用 PB6.0 建立分布式应用包括客户机端和服务器端的功能设计。客户机端的用户接口和服务器端的数据服务部分与传统的 C/S 应用中相应的功能基本一致，而商业逻辑的包装、配置及协调等则是分布式应用开发的关键。

1.商业逻辑的配置原则

PB6.0 通过对象体系结构的逻辑扩展，提供了不可视的定制类用户对象来包装商业逻辑，从而支持应用的分布。实际上，包装好的商业逻辑需要配置到不同的机器上。把所有的商业逻辑对象都放在服务器端并不是最佳的选择，这可能会降低网络和服务器的性能。因此，配置商业逻辑对象可参考如下几个原则：

(1)将对象放在客户机应用中的情况

·商业逻辑对象非常稳定

·对象不依赖于服务器维护的数据

·时间上需要立即响应

(2)将对象放在服务器应用中的情况

·商业逻辑对象被许多应用使用或需频繁地更改

·对象含有敏感信息

·对象需要异种数据源的数据

商业逻辑对象的位置确定之后,就可以进行客户机端和服务器端的应用设计了,下面结合例子进一步说明。

2.客户机应用设计

客户机应用包括窗口、菜单和按钮等可视对象及其响应事件的脚本设计,这和一般的应用情况类似。除此以外,重点要实现与服务器的连接和调用远程商业逻辑对象等功能。

(1)连接服务器。客户机应用使用不可视的连接对象连接到服务器。由于连接对象不是预制的全局对象,所以必须在应用代码中说明一个全局的或实例的连接类型的变量。建立连接的步骤如下:

·用 Create 语句实例化连接对象(Connection object)

·设置连接对象的属性

·调用 ConnectToServer 函数建立到服务器应用的连接

·进行必要的错误检查和处理

以下是建立连接的脚本示例:

```
//Global variable:
//connection myconnect
long ll-rc
myconnect = create connection
myconnect.driver = "Winsock"
myconnect.application = "dpbserv"
myconnect.location = "server01"
ll-rc = myconnect.ConnectToServer()
IF ll-rc<>0 THEN
  MessageBox("Connection failed", ll-rc)
END IF
```

其中,Winsock 是选用的通信驱动程序,dpbserv 和 server01 分别是服务器应用名和服务器名,它们应当在 host 文件和 services 文件中说明。或者直接在连接对象的属性中使用端口号和 IP 地址。

(2)调用远程对象函数。建立连接后就可以调用服务器上商业逻辑对象的函数和访问对象的实例变量。调用远程对象的函数要执行下列脚本语句:

·用 CreateInstance 函数实例化远程对象(Remote object)

·调用该远程对象的函数

下面是实例化服务器上的远程对象和调用其函数的脚本示例:

```
//Global variable:
//connection myconnect
uo-customer iuo-customer
string ls-custid
long ll-rc
ls-custid = Trim(sle-custid.text)
ll-rc = myconnect.CreateInstance(iuo-customer)
IF ll-rc<>0 THEN
  MessageBox("CreateInstance failed", ll-rc)
END IF
IF iuo-customer.retrieve-balabce(ls-custid) != 1 THEN
  MessageBox("Error","Retrieve failed")
END IF
```

由于 PB6.0 提供了位置透明性,远程对象类型的变量说明可以拥有本地对象实例的引用或远程对象实例的引用,客户端脚本不需要知道该对象在何处创建。只是在对象实例化时才考虑其物理位置。运行时可以根据应用需求,进行对象的本地实例化(用 Create 语句)或远程实例化(用 CreateInstance 函数),前者在客户机上需要有类的完整实现,后者仅需要代理对象(Proxy object)即可。如上述代码中的 uo-customer 是服务器上远程对象的代理对象,而 iuo-customer 是远程对象在客户机上的一个实例。

(3)定义远程对象同步方式。PB6.0 提供同步调用和异步调用两种方式,实现客户机应用和服务器应用之间远程对象的同步。客户机使用同步调用时,服务器立即执行远程对象函数,客户机一直等到服务器处理完成后才继续执行其他脚本语句。客户机使用异步调用时,服务器将请求加入到队列中,在以后的某个时刻执行并通过推送技术通知客户机;此时客户机不必等待,可以继续执行其他脚本语句。同步调用使用 TRIGGER 关键字,异步调用使用 POST 关键字。

3.服务器应用设计

服务器应用的设计主要包括管理客户机连接、包装商业逻辑和访问数据库等功能。

(1)管理客户机连接。应用程序只需一个窗口来启动和停止侦听及监视客户机连接。与客户机建立连接的代码类似,服务器需要在脚本中执行下面的语句:

·用 Create 语句实例化传输对象(Transport object)

·设置传输对象的通信驱动和服务器应用属性

·调用传输对象的侦听函数 Listen()捕获客户机连接请求

·在连接开始(ConnectionBegin)事件中返回连接的特权值

·相关的错误处理

为了访问数据库,可在 ConnectionBegin 和 ConnectionEnd 事件中分别包含连接和断开数据库的命令。

(2)包装商业逻辑。在 PB 中用远程对象的方法来包装应用的商业逻辑,方法的功能由它定义的函数实现,函数描述了逻辑规则,这与一般应用中逻辑规则的设计方法相同。远程对象是不可视的定制类用户对象,其方法的函数只可使用标准数据类型、结构及结构数组和定制类用户对象作为变元,可视对象或预定义的不可视对象(如事务对象,数据存储对象)不能用做方法的变元。方法的返回值可以是标准数据类型、结构或定制类用户对象。

(3)访问数据库。服务器端访问数据库一般采用不可视的数据窗控件,即数据存储(DataStores)对象。以检索操作为例,对象脚本一般执行下述语句:

·创建一个数据存储对象的实例并分配一个数据窗对象

·为数据存储设置数据库事务对象

·执行数据存储对象的检索函数 Retrieve()

·对结果集进行相应的处理

下面是检索数据并将结果传回客户机的函数代码示例:

```
//customers 是一个结构数组变元,结构与数据窗 d-custlist 布局相同
datastore ds-datastore
long ll-rowcount
ds-datastore = create datastore
ds-datastore.dataobject = "d-custlist"
ds-datastore.SetTransObject(SQLCA)
IF ds-datastore.Retrieve() <> -1 THEN
  ll-rowcount = ds-datastore.RowCount()
END IF
Customers = ds-datastore.object.data
Destroy ds-datastore
Return ll-rowcount
```

为了使客户机上的数据和服务器上的数据保持一致,必须执行移动数据窗缓存和状态标志信息的操作,从而保持数据存储和数据窗控件数据更新的同步化。

(4)向客户机推送消息。当客户机采用异步调用时,服务器利用推送技术向客户机发送消息,通知它异步请求已完成。客户端对象的异步请求先进行排队,并在所有的同步请求之后处理。为了发送消息,服务器需要知道发送给哪一个客户端的对象。所以,客户机必须传递一个不可视对象的引用给服务器,服务器接收对象引用后自动产生一个客户端对象的远程引用,并调用与这个对象关联的函数,函数调用通过网络传回到含有那个对象的客户机。

五、结束语

在分布式应用设计中,重点是合理地划分商业逻辑并创建其对象和方法,根据应用规模和对象性质在应用服务器上分布。同时还要处理好客户机与服务器之间的同步问题,以及防止由于组件分布和共享而可能产生系统死锁等。除了支持基于 Client/Server 的分布式应用,PB 还提供 Web.PB 使得应用可以在 Internet 上分布,通过调用远程对象的服务可以为客户端的浏览器提供动态内容。

参考文献

[1] 微软公司,"Microsoft Tech - Ed 98 资料",吉林科学技术出版社,1998.7

[2] Sybase 公司,"PowerBuilder Version 6.0 Online Books"

[3] Simon Gallagher & Simon Herbert 著,康博创作室译,"PowerBuilder 6.0 程序设计大全",机械工业出版社,1998.8

(来稿时间:1999 年 3 月)