利用 PowerBuilder 开发分布式组件及实例

田生伟 (乌鲁木齐新疆大学计算机系 830046)

摘要： 本文探讨了PowerBuilder分布式应用程序的特点和机制，并给出了实例，总结了开发经验。

关键词： 客户/服务器 分布式 组件 远程对象 代理对象

1 序言

早期的客户/服务器结构基于双层结构，一层是客户层，另外一层是服务器层，这种结构存在中心控制困难，安全性较差，客户端负载沉重等缺陷。为了克服这些缺陷，可以引入一种改进的客户/服务器配置方案，即三层客户/服务器体系：

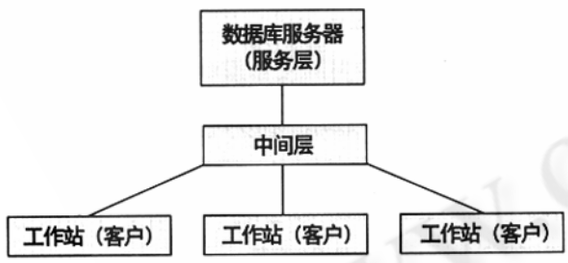

图 1 三层客户／服务器体系

在这个方案中有三个逻辑层，客户层面向用户，服务器层面向数据服务，而中间层面向商业或企业规则。这种分布式客户/服务器结构强调组件开发，将原来许多客户端的处理分离出来，形成相对独立的组件(中间件)放在服务器上，供所有的客户端应用程序访问。POWERBUILDER使用Jaguar CTS(组件事务服务器)、MTS(微软事务服务器)和分布式PowerBuilder技术提供对三层模式应用的支持，允许开发者创建中间层Jaguar CTS、MTS和分布式PowerBuilder服务器上运行的组件(或对象)，并在客户端调用这些组件，从而得到服务器提供的服务。

2 分布式 PowerBuilder 应用的基本结构

2.1 服务器应用程序

服务器应用程序有两个主要组件：传送对象(Transport)和远程对象。传送对象能使服务器接收客户连接并处理客户提出的服务请求；而远程对象是一个定制类用户对象，包含在服务器的应用程序中，可以被客户应用程序调用。

2.2 客户应用程序

客户应用程序有三种组件：用户接口、连接对象(Connection)和远程对象类的定义。用户接口指与用户进行交互的窗口、菜单等；连接对象(Connection)使客户应用程序与服务器建立连接，保证客户应用程序利用服务器的服务。服务器应用程序中的远程对象在客户应用程序中有一个与之对应的代理(Proxy)对象。代理对象是由服务器生成的，在开发期间代理对象为开发客户端应用程序提供了完整的类定义，客户应用程序编写完毕后，可以通过代理对象执行远程服务器中的相应程序。

3 实例

本实例是一个运行在服务器上的监听登录客户系统，可以监听客户的登录、使用状态、退出时间等信息；同时，客户应用程序调用远程对象来对数据库查询和更新。

3.1 启动服务器程序

为了启动服务器监控程序，应说明并实例化Transport对象，用Listen函数启动来自客户端的连接，通过ConnectionIfo对象和GetServerInfo获取连接到服务器

的客户端的信息。

为此，先定义全局变量：

```
Transport v_transport
```

在启动服务器监控程序窗口的Open事件写下脚本：

```
long v_receiver
v_transport=creat transport // 创建 Transport 对象实例
v_transport.driver="winsock" // 声明通信驱动程序
v_transport.application="6000" // 用端口号指定应用程序名
v_transport.location="xjunt" // 指定计算机位置
v_transport.options=""
v_receiver=v_transport.listen() //启动对客户端的监听
if v_receiver<>0 then
messagebox("提示", "启动错误")
else
messagebox("提示", "正常启动")
end if
```

3.2 客户应用程序连接服务器

为了客户端程序连接到服务器，应说明并实例化Connection对象，建立客户端到服务器应用的连接，再用ConnectToServer 函数连接到服务器。在客户应用程序中，首先定义全局变量：

```
Connection v_connect
```

在"登录"(Command Button)的Click事件写下脚本：

```
long v_receiver
v_connect=create connection // 创建 connection 对象实例
v_connect.driver="WinSock" // 声明通信驱动程序
v_connect.application="6000" //用端口号指定应用程序名
v_connect.location="xjunt" // 指定连接服务器位置
v_connect.options=""
v_connect.userID=sle_id.text // 得到客户帐号
v_connect.connectstring=sle_name.text //得到客户名
v_receiver=v_connect.connectToserver() // 连接服务器
if v_receiver<>0 then Messagebox("提示", "登录失败")
```

3.3 获得客户信息

系统管理员拥有权限查看登录到服务器的客户端的信息，可以在服务器应用程序的ConnectionBegin事件写下脚本：

```
If userid="sa" and password="system" then
return ConnectWithAdminPrivilege! //
Else
return Connectprivilege! //
End if
```

在服务器应用程序的窗口的 Open 事件中补充下列脚本：

```
v_connect.userid="sa"
v_connect.password="system"
v_receiver=v_connect.connectToServer()
timer(2) // 每两秒激活 Timer 事件检测客户信息。
```

在服务器应用程序的窗口的Timer事件中写下脚本：

```
Connectioninfo guest [ ]
Int v_rownum
Long v_rows
V_rows=v_connect.GetServerInfo(guest)
Dw_1.reset()
For v_rownum=1 to v_rows
If guest [v_rownum].ConnectUserId<>"sa" then
Dw_1.inserrow(0)
Dw_1.setitem(v_rownum-1,1,guest [v_rownum].ConnectUserID)
Dw_1.setitem(v_rownum-1,2,guest [v_rownum].ConnectString)
Dw_1.setitem(v_rownum-1,3,guest [v_rownum].Connecttime)
End if
Next
```

3.4 建立代理对象

中间件存在于服务器的计算机上，是不可视的类用户对象，在服务器上执行，为客户应用程序调用，所以称远程对象。为了使客户端的程序能够调用远程对象，应为客户程序生成代理对象，以便客户通过代理对象访问在服务器上的远程对象。生成代理对象的过程如下：点击画板的NEW 按钮选择 PROJECT 中的 ProxyLibrary,打开 Project 画板的 Select Objects 对话框，选择要生成的代理对象，待选择用户对象后，返回Project画板。在Project画板的 Properties for Proxy Library 对话框为生成的代理库指定属性，为生成的代理库指定库文件的路径和名称。

3.5 数据库的访问

客户端的应用程序可以通过数据窗口访问数据库，但所有的处理操作均是在服务器端的数据存储对象中进行，通过服务器端的数据存储对象对数据库操作(查询、更新等)。

为了实现利用远程对象在服务器上对数据库的操作，可在服务器应用程序中定义类用户对象 uo_guest：

声明实例变量：

datastore ds_guest

在 Constructor 事件中写下脚本：

ds_guest=creat datastore

ds_guest.dataobject="d_guest"

ds_guest.settransobject(sqlca)

编写一个数据查询的对象函数：retrieve_da(ref blob v_blda)returns long

long v_rows

ds_guest.retrieve() // 把数据检索到数据存储仓库 ds_guest 上

v_rows=ds_guest.getfullstate(v_blda) //得到数据存储仓库的完整状态存在 v_blda 中 return v_rows // 返回记录条数

编写一个数据更新的对象函数：update_da(ref blob v_blda) returns long

long v-state

if ds_guest.setchanges(v_blda)=1 then

v_state=da_guest.update()

end if

if v_state=1 then

commit;

ds_guest.getchanges(v_blda)

else

rollback;

end if

return v_state

然后为 uo_guest 生成代理对象。对客户应用程序添加脚本：

uo_guest uo_inguest // 说明实例变量

在“查询”(Command button)的 Click 事件中写下：

Blob v_blob

Int v_rows

v_connect.createInstance(uo_inguest)//建立远程对象实例

v_rows=uo_guest.retrieve_da(v_blob)

If v_rows>=0 then

Dw_1.setfullstate(v_blob) //得到数据窗口的完整状态存在 v_blob 中

Else

Messagebox(“提示”，“查询失败”)

End if

在“更新”(Command button)的 Click 事件中写下：

Blob v_upblob

Long_v_upstate

v_upstate=dw_1.getchanges(v_upblob) // 得到完整修改操作存在 v_upblob 中

if v_upstate=-1

Messagebox(“提示”，“更新失败”)

Else

If uo_inguest.update_da(v_upblob)then dw_1.setchanges(v_upblob) // 调用远程对象修改，并将结果返回客户应用。

4 总结

利用 Powerbuilder 开发分布式应用，可以进行组件的设计和实施，供不同的应用程序共享，提高了复用性；把组件安排在客户工作站以外的机器上，能将处理负担转移到功能强大的服务器上，均衡负担，提高了资源利用率；将应用程序的功能封装到不同的组件，便于管理，也更利于维护。■

参考文献

1 [美] JOE SALEMI，客户／服务器数据库指南，电子工业出版社。

2 [美] SteveErlank，Craig Levin。

3 sybooks.Sybase.com。

4 www.sybase.com。

5 support.Sybase.com。