基于.NET Remoting 的 CNG 管理系统的设计与实现①

Design and Implementation of CNG Management System Based on .NET Remoting

王 翔[1] 陈蜀宇[2] (1.重庆大学 计算机学院 重庆 400030; 2.重庆大学 软件学院 重庆 400030)

摘 要: 介绍了.NET Remoting 的体系结构和原理。通过在 CNG 管理系统中的应用为例，展示如何使用.NET Remoting 构建安全的分布式应用系统，提高代码的灵活性和可扩展性。最后给出了用 C#.NET 实现的具体方法和步骤。

关键词: .NET Remoting CNG 安全 分布式应用系统 C#.NET

随着网络技术的发展，分布式计算已经成了软件开发中不可或缺的部分。远程访问的方式主要有三种：Microsoft 的 COM/DCOM 技术、SUN 的 EJB 技术和 OMG 的 COBRA 技术。DCOM 技术具有语言无关性，并且能非常方便地建立可伸缩的应用系统，得到了广泛的应用，但其在部署方面较为复杂，不同版本必须保持一致，否则会出现“动态链接库地狱”问题。

.NET 技术是微软公司推出的下一代平台技术，其中以.NET Remoting 取代以往的 DCOM 技术，成功地解决了以上的问题。.NET Remoting 是一个内容丰富的、可扩展的框架，它使得分布在不同应用程序域(AppDomain)、不同过程和不同计算机上的对象可以实现无缝通信。.NET Remoting 提供的编程模型和运行时支持功能强大而又易于使用，能够实现透明的交互。

本文将介绍.NET Remoting 的体系结构和原理，并以在 CNG(Compressed Natural GAS,压缩天然气)管理系统中的应用为例，介绍实现的过程。

1 .NET Remoting 分布式应用体系结构及原理

.NET Remoting 中通过通道（channel）来实现两个应用程序域之间的对象通信。Remoting 的通道主要有两种：Tcp 和 Http。在.NET 中，System.Runtime.Remoting.Channel 中定义了 IChannel 接口。IChannel 接口包括了 Tcp 通道类型和 Http 通道类型，它们分别对应 Remoting 通道的两种通道类型。通道对象代表了到远程应用程序的连接。每个通道对象还包含格式化程序对象，将方法调用转换为已知格式的消息。

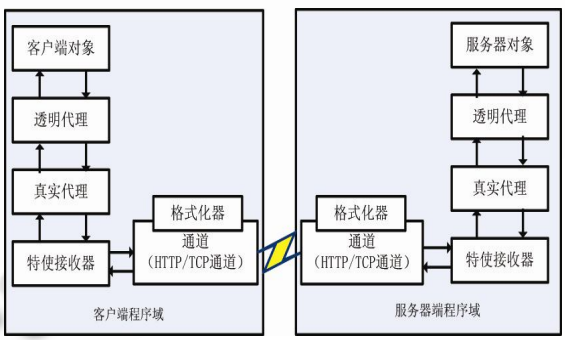

图 1 .NET Remoting 体系结构图

其基本原理如图 1 所示。首先，客户端通过通道访问服务器端对象，以获得服务器端对象的代理。服务器端对象也即远程对象，使用时是通过跨应用程序边界传递对象引用获得该远程对象的代理。对于客户程序来说，代理提供了与远程对象完全一样的方法和属性。当代理的方法被调用时，就会创建消息，通过使用格式化程序类，将这些消息串行化并发送到客户通道中。客户通道和服务器通道进行通信，以通过网络传输消息。服务器通道则使用格式化程序并行化消息，从而将方法发

① 基金项目:新世纪优秀人才支持计划(教技函[2005]35 号 NCET-04-0843)
收稿时间:2008-10-24

送给远程对象。通过代理，客户端应用程序就可以像使用本地对象一样来操作远程对象。

2　.NET Remoting在CNG管理系统中的应用

重庆市推广应用天然气汽车已 7 年多，天然气汽车保有量达 4 万余辆、天然气加气站(含在建)共 61 座、气瓶 5 万多支，该管理系统的建立，就是为了达到汽车行业的天然气管理和消费信息化，以加强对天然气汽车加气的安全管理，改变当前天然气管理困难的局面。

2.1 CNG 管理系统的系统结构

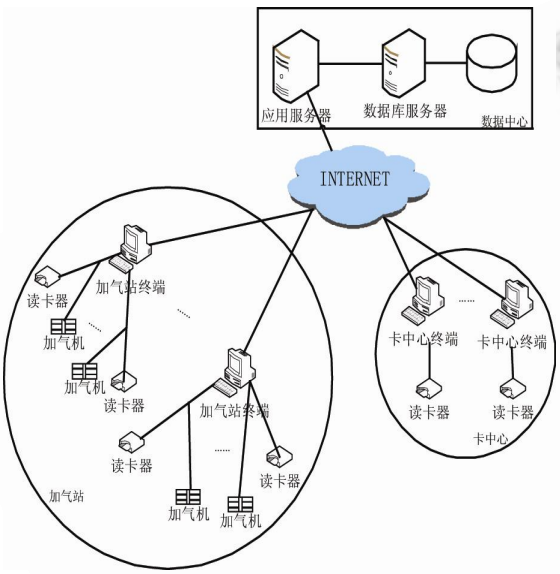

图 2　CNG 管理系统结构图

CNG 管理系统的系统结构如图 2 所示，主要分为以下几个主要组成部分：

2.1.1 数据中心应用服务器及数据库服务器

数据中心应用服务器主要运行管理系统服务器端，处理来自各个客户端的业务请求，例如：处理来自加气站客户端的刷卡消费、加气费率设置，卡中心的办卡、注销卡等请求，数据库服务器运行 ORACLE 10G，以供服务器端存取数据。

2.1.2 加气站终端

加气站终端使用 RS-232 总线(采用分线器)连接所有的加气机和 IC 卡读卡器，使用读卡器对司机出示的 IC 卡进行验证，通过验证则自动控制打开/关闭加气枪，业务完成后加气站客户端将会自动将该笔业务上传到服务器进行处理。在网络不通或者是服务器忙的情况下采取本地暂存业务记录的方式，网络一旦通畅，即刻上传缓存的业务数据。

2.1.3 卡中心终端

司机必须先办卡才能持卡进行加气，卡中心主要处理办卡和挂失等类型的卡业务，业务信息由卡中心客户端上传到服务器。

2.1.4 读卡器和加气机

加气站终端和卡中心终端都使用读卡器，以向 IC 卡读取和写入 IC 卡信息。加气机只有在加气站使用，由加气站客户端根据 IC 卡的信息来控制气枪的打开和关闭。

2.2 系统交互过程

所有的客户端都是通过 INTERNET 和服务器进行数据的传输。系统交互的过程为：在服务器中的指定端口发布.NET Remoting 远程对象，客户端为了上传业务信息远程获得服务器发布的对象，以业务数据包装类的对象作为参数调用服务器对象中的方法，并通过 DATA TABLE 或者是其他类型的数据作为方法的返回值。下面分别介绍业务数据包装类、服务器和客户端的实现。

2.3 业务数据包装类的实现

客户机向服务器传输的都是业务记录，每一条记录都对应着服务器数据库的若干张表。现对数据库中的所有表进行类化，在客户端向服务器上传数据的时候使用这些包装类的对象进行传输。由于是在网络上进行传输，故要给表类设置序列化属性。下面以加气业务(GASBUSS)表的包装类为例进行演示：

```
[Serializable]//序列化属性
public class GASBUSS
{
  public GASBUSS()//构造函数
  {.....}
  public int GASBID;//加气站 ID
  public string ICID;//IC 卡 ID
  public string GASBUSSID;//加气业务 ID
  ......//其他的字段
}
```

2.4 服务器端的实现

按照所有客户端所需要进行调用的功能来编写类，最后编译成 ServerFunctions.dll 文件，以供其他的程序引用。服务器端通过读取配置文件的方式发布远程对象，并打开指定的端口进行监听，以便客户端获取远程对象并进行远程调用。由于客户端的类型分为加气站和卡中心，故编写两个类，分别封装不同的客户端会调用的函数，类的名称分别为 GasStation 和 CardCenter。

2.4.1 远程类的实现

远程类中封装了一系列的业务处理函数。现以一个业务函数：提交消费记录的函数为例进行说明，异常处理代码略去：

```
public int inputBusiness(GASBUSS NewBuss)
{
  ......//一系列的合法性验证语句，若不合法则返回 INT 类型的出错代码
  ......//OracleCommand 和 OracleTransaction 对象的初始化，准备处理业务
  string sql="INSERT INTO GASBUSS(GASBID,ICID,GGID,GASBUSSID,GASNUM,GASPRICE,MASTTOTAL,REALTOTAL,ADDTIME)VALUES("+"SEQ_GASBUSS_ID.Nextval,'"+NewBuss.ICID+"','"+NewBuss.GGID+"','"+NewBuss.GASBUSSID+"','"+NewBuss.GASNUM+"','"+ NewBuss.GASPRICE+"','"+NewBuss.MASTTOTAL+"','"+NewBuss.REALTOTAL+"',to_date('"+NewBuss.ADDTIME.ToString()+"','YYYY-MM-DDHH24-MI-SS')")";//根据业务数据拼装 SQL 语句
    myOracleCommand.CommandText = sql;
    myOracleCommand.ExecuteNonQuery();
    ......//其余的 SQL 业务语句
    myOracleTransaction.Commit();
    return 1;
}
```

2.4.2 服务器的配置文件 App.config

```
<configuration>
  <system.runtime.remoting>
    <application name="RemoteServer">
      <service>
        <wellknown
type="ServerFunctions.GasStation,ServerFunctions"
objectUri="ServerFunctions.GasStation"
mode="SingleCall" />
        <wellknown
type="ServerFunctions.CardCenter,ServerFunctions"
objectUri="ServerFunctions.CardCenter"
mode="SingleCall" />
      </service>
      <channels>
        <channel ref="tcp" port="8086"/>
      </channels>
    </application>
  </system.runtime.remoting>
</configuration>
```

.NET 的配置文件采用 XML 格式。最外层的元素是<configuration>，这是所有配置文件所必须有的。所有的远程配置项必须作为子元素添加到<system.runtime.remoting>下面。<application>元素使用 name 属性指定了服务器的名称。应用程序所提供的服务必须作为<service>的子元素列出，这就是远程对象本身，可以使用<wellknown>元素来指定远程对象，mode 属性可以指定为 SingleCall 或 SingleTon，SingleCall 是一种无状态模式，一旦设置为该种模式，当客户端调用远程对象的方法时，Remoting 会为每一个客户端建立一个远程对象实例，SingleTon 为有状态模式，为所有客户端建立同一个对象实例。同时用 type 属性来指定已经定义的类的类名。在<channels>元素中，定义了服务器要使用的通道，用 ref 属性可以引用一个预先定义好的通道，同时必须使用 port 属性为通道分配端口。

2.4.3 远程对象的发布

远程对象的发布可以使用多种方式进行承载，比如 WINFORM，CONSOLE 或者是 WINDOWS SERVICE。由于不需要进行界面显示，并且要方便查看运行状态，故 CONSOLE 较为合适。在 CONSOLE

的 MAIN 函数中读取服务器端配置文件，就实现了远程对象的发布：

```
RemotingConfiguration.Configure("App.config", false);
```

2.5 客户端的实现

CNG 管理系统客户端包括加气站客户端和卡中心客户端。下面以加气站的客户端的实现为例进行说明。

2.5.1 客户端引用远程对象

为了实现与远程对象的通信，客户端程序必须添加对 System.Runtime.Remoting.dll 和前面创建的 ServerFunctions.dll 程序集的引用。

2.5.2 客户端的配置文件

```
<configuration>
  <appSettings>
    <add key="ServiceURL" value="tcp://219.153.2.46:8086/ServerFunctions.GasStation"/>
  </appSettings>
</configuration>
```

客户端的配置文件和服务器端的配置文件大致相同，只是没有端口的配置。key 元素的键值对应着服务器发布的对象名称，包括四个部分：通道类型、IP 地址、端口号、类名。

2.5.3 客户端获取服务器端的远程对象

下面以充气的业务提交函数为例说明客户端是如何获得远程对象和调用远程对象中的方法的：

```
public int inputBusiness(DateTime mydate)
{
    GASBUSS NewBuss = newGASBUSS();//申明表类的对象，使用这个对象承载业务数据
    NewBuss.ICID = icID;
    NewBuss.GASBUSSID = gasID + "_" + icID + "_" + mydate.ToString();
    ......//其他业务数据项的设置
    GasStation mgs=(GasStation) Activator . GetObject(typeof(GasStation),System.Configuration.ConfigurationManager.AppSettings["ServiceURL"]);//获得远程对象，ServiceUrl 的键值对应着服务器发布的对象名
  int Result = -1;
  try
  {
    if ((InputLocalBussToDB(NewBuss,0)) ! = 1) throw new Exception("Local saving fail"); //InputLocalBussToDB 为本地函数，本地操作如果失败马上抛出异常
    if ((Result = mgs.inputBusiness(NewBuss))==1)//调用远程对象方法，如果失败马上抛出异常
    Result = 1;
    else{Result=2;throw new Exception ("Remote saving fail");}//远程调用产生异常
    }
    catch(Exception e)//在网络不通的情况下{ .....//异常处理代码}
  return Result;
}
```

3 改进的安全配置

该系统运行在广域网的环境下，其涉及到金额流转，对于金额业务数据的加密和对客户端的合法性认证成为了该系统改进的重点，.NET Remoting 中可以通过配置的方式轻易地实现数据的加密和客户端的认证。只要对服务器端和客户端的配置文件稍加修改即可实现。

在客户端配置文件中的 configuration 属性内添加以下配置文件：

```
<system.runtime.remoting>
  <application name="RemoteClient">
    <channels>
      <channel ref = "tcp" secure ="true" impersonationLevel="Impersonation" protectionLevel="EncryptAndSign" username=" CngUser " password="******"/>
    </channels>
  </application>
</system.runtime.remoting>
```

(下转第 106 页)

(上接第 8 页)

并将服务器配置文件的 channel 属性修改成与更改后的客户端配置中的 channel 属性一致即可。

其中，channel 的 secure 设置为 true 表示打开 Remoting 的传输安全性，impersonationLevel 设置为 Impersonation 表示客户端以 username 的身份来进行远程调用，protectionLevel 设置为 EncryptAndSign 表示传输的数据均要加密并带有签名，username 和 password 表示要进行远程调用的时候所需要进行验证的用户名和密码。

在对配置文件进行了以上的修改并经过重新编译并运行，通过网络抓包软件 WireShark 截取数据包，服务器以及客户端之间交换的数据均以加密的方式出现在网络中。虽然服务器已经对外公布了服务名，但如果没有正确用户名和密码，就不能获得远程对象以进行远程调用，通过这种方式对客户端进行认证。

4 结论

本文讨论了.NET Remoting 在重庆市 CNG 管理系统中的应用，该系统已经投入了一期的试用阶段，具有一定的使用人群，服务器端和客户端均能正常工作，网络通信延迟较短。使用.NET Remoting，确实将开发人员的工作重心转移到了业务流程中来，极大地缩短系统的开发时间和减小开发难度。其提供的传输加密和远程调用端认证的功能，也为该系统的开发提供了极大的便利。

参考文献

1 Nagel C et al. Professional.Wrox Press 2006.

2 Robinson S,Allen K.杨浩,杨铁男译.C#高级编程.北京:清华大学出版社, 2002.

3 Rammer I. Advanced .NET Remoting Springer- Verlag New York Inc,2002

4 王常力,罗安.分布式控制系统(DCS)设计与应用实例.北京:电子工业出版社, 2004.