运用CAR智能指针实现Callback机制

叶 蓉[1,2],陈 榕[1]
(1. 同济大学 基础软件工程中心,上海 200092;
2. 上海第二工业大学,上海 201209)

摘 要:"和欣"操作系统是基于CAR构件技术、支持构件化应用的嵌入式操作系统。一般的构件,客户与构件之间的通信过程是单向的,客户创建构件对象,然后客户调用对象所提供的接口函数。在这样的通讯过程中,客户总是主动的,而构件对象则处于被动状态。对于一个全面的交互过程来说,这样的单向通信往往不能满足实际的需要,构件对象也要主动与客户进行通信,构件也提供回调接口。和欣系统中的Callback机制有助于实现二进制构件拼装;并允许构件异地运行,可极大地提高构件的运行效率,但其本身实施过程很复杂。提出在"和欣"操作系统中,实现CAR智能指针来简化用户实现Callback机制的过程。

关键词:CAR;CAR智能指针;Callback机制
中图分类号:TP311 **文献标识码**:A **文章编号**:1673-629X(2008)02-0009-04

Using CAR Smart Pointer to Realize Callback Mechanism

YE Rong[1,2], CHEN Rong[1]
(1. System Software Engineering Centre of Tongji University, Shanghai 200092, China;
2. Shanghai Second Polytechnic University, Shanghai 201209, China)

Abstract: Elastos operating system, which can sustain component applications, is based on component assembly runtime technique. Communication process between normal component and client is one-way oriented, client builds component object, and then calls interface provided by the object. In this communication process, the client is always active, and component object is in a passive state. For a comprehensive interactive process, such one-way communication often cannot meet the actual needs, component object also need take the initiative to communicate with client, as well as to provide call back interface. Elastos Callback mechanism is helpful for binary components' integration and it allows components running in different context, which largely improves the CAR component running efficiency. However the normal Callback implementation is too much complicated during execution. Depicts how to use CAR smart pointer to simplize the application procedure of Callback mechanism.

Key words: component assembly runtime; CAR smart pointer; Callback mechanism

0 引 言

基于构件的软件工程是实现高生产率、低维护费用和高可靠软件产品的关键技术。构件的使用方式类似于"客户/服务器"模型,其中构件充当服务器的角色。

一般的构件,客户与构件之间的通信过程是单向的,客户创建构件对象,然后客户调用对象所提供的接口函数。在这样的通讯过程中,客户总是主动的,而构件对象则处于被动状态。对于一个全面的构件交互过程来说,这样的单向通信往往不能满足实际的需要,有时候构件对象也要主动与客户进行通信,因此,与普通接口(interface,也称为入接口)相对应,构件也可以提供回调接口(icallback,也称为出接口),对象通过回调接口与客户进行通信。Callback机制有助于实现二进制构件拼装;并允许构件异地运行。极大地提高了构件的运行效率,但Callback机制实施过程很复杂,因此采用类智能指针包装其交互过程来简化用户使用的复杂性。

文中提出在CAR构件技术中运用智能指针对客户端进行简化,以方便用户使用CAR构件。用户可以通过宏定义选择直接通过接口指针进行操作,亦可以选择通过智能指针完成对接口的使用。CAR的智能

收稿日期:2007-05-31
基金项目:国家"863"计划资助项目(2001AA113400)
作者简介:叶 蓉(1980-),女,江苏南京人,硕士研究生,研究方向为嵌入式操作系统、系统软件支撑技术;陈 榕,教授,博士生导师,研究方向为嵌入式系统、构件技术。

指针可以向用户屏蔽了对构件生命周期的管理，用户使用智能指针，无需再去考虑繁琐复杂的 AddRef、Release 的调用，简化了编程的步骤，降低编程难度，同时确保客户能正确地控制构件的生命周期，提高了用户应用程序的安全性，降低了资源泄漏的可能；通过智能指针能够检查接口的类型安全性，消除潜在的错误。

1 “和欣”操作系统和 CAR 构件编程环境

1.1 “和欣”操作系统

“和欣”[1]操作系统是 863 计划的“基于中间件技术的因特网嵌入式操作系统及跨操作系统中间件运行平台”的重要成果，是一个基于构件的灵活内核现代操作系统。“和欣”操作系统与宏内核或微内核操作系统的最大区别就是其将微内核模型与基于构件技术的充分结合，形成了“和欣”操作系统的灵活内核架构模型。

1.2 CAR 构件程序集运行时

CAR(Component Assembly Runtime)是国内拥有自主知识产权的先进构件系统。CAR 构件技术定义了一套网络编程时代的构件编程模型和编程规范，它规定了一组构件间相互调用的标准，使得二进制构件能够自描述，能够在运行时动态链接。CAR 构件技术继承了 COM[2,3]的二进制封装思想，面向接口编程。逐步融合.Net、Java 技术思想之后，形成独有的二进构件程序模型。CAR 构件技术采用 C++编程，使和欣 SDK 提供的工具直接生成运行于和欣构件运平台的二进制代码。因此 CAR 构件机制使得程序能够充分运用自己熟悉的编程语言知识和开发经很容易掌握面向构件、中间件编程的技术。

用智能指针实现 CAR 回调机制

回调机制

构件模块之间总是存在着的接口，从调用方式上，可为三类：同步调用、回调和用。同步调用是一种阻用，调用方要等待对方毕才返回，它是一种单；回调是一种双向调用就是说，被调用方在接用时也会调用对方的接调用是一种类似消息机制，不过它的调用相反，接口的服务在讯息或发生某种事件时，会主动通知被调用方(即调用被调用方的接口)。回调和异步调用的关系非常紧密，通常使用回调来实现异步消息的注册，通过异步调用来实现消息的通知。同步调用是三者当中最简单的，而回调又常常是异步调用的基础。

具体来说，回调如图 1 所示，就是模块 A 调用模块 B，而模块 B 中要调用模块 A 中的函数 C 的代码，其中函数 C 就是回调函数；

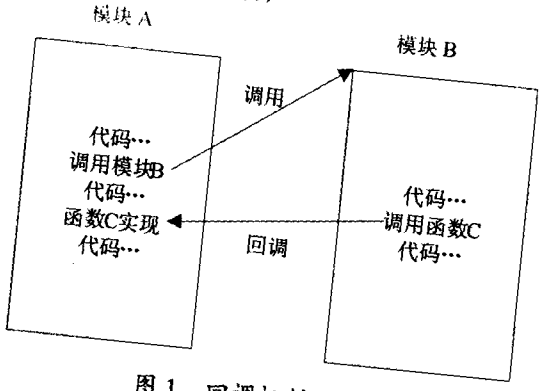

图 1 回调机制的实现

客户端(模块 A 地址空间)首先通过同步方式调用服务端(模块 B 地址空间)的注册接口来注册回调接口，服务端收到该请求以后，就会保留该接口引用，如果发生某种事件需要向客户端通知的时候就通过该引用调用客户方的回调函数，以便对方及时处理。

在 CAR 的回调机制中，有一种接收器(sink)对象，接收器对象相当于一个客户端回调函数的容器，在客户端的地址空间里，负责与可连接对象进行通信。只要有可连接对象存在，那么在客户端肯定要有接收器的存在。多个回调接口可对应一个接收器对象，这样可以减少通信花费的开销。当接收器与可连接对象建立连接后，客户程序可将自己实现的事件处理函数(回调函数)向接收器进行注册，而不是向可连接对象进行

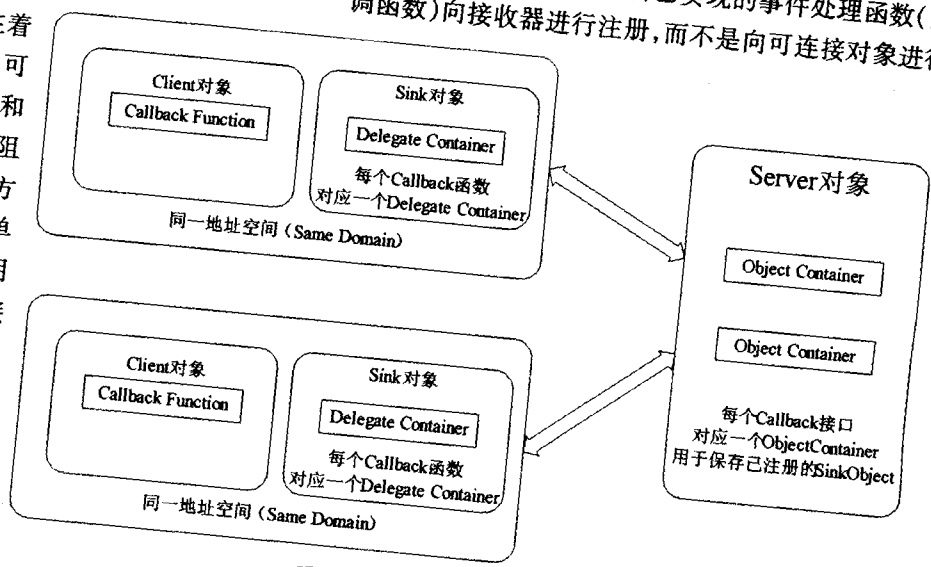

图 2 CAR 回调机制具体实现

注册,又一次减少了通信开销。接收器会自动把在它里面注册的函数的回调接口指针告诉构件对象,构件对象在条件成熟时激发事件,如果客户注册了自己的回调接口方法,那么就会被调用,否则就调用接收器默认实现的回调接口方法。其过程如图2所示。

在编写构件程序时,用户需定义何时激发事件,在编写客户端程序时,用户需在适当的时候注册事件处理函数。其它的工作,如接收器对象的实现、接收器与可连接对象建立通信的具体过程、事件的分发回调过程等都由CAR实现。

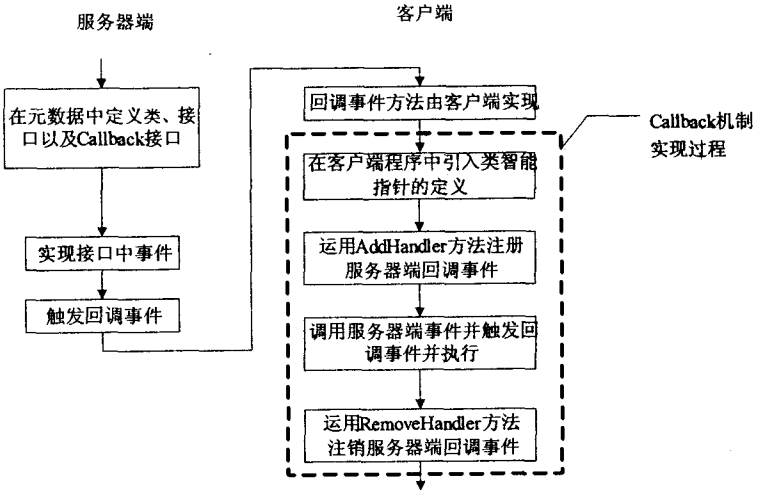

图4　使用CAR智能指针的Callback机制实现

2.2　未采用CAR智能指针实现回调(Callback)机制

未使用CAR智能指针的回调(Callback)过程如图3所示。

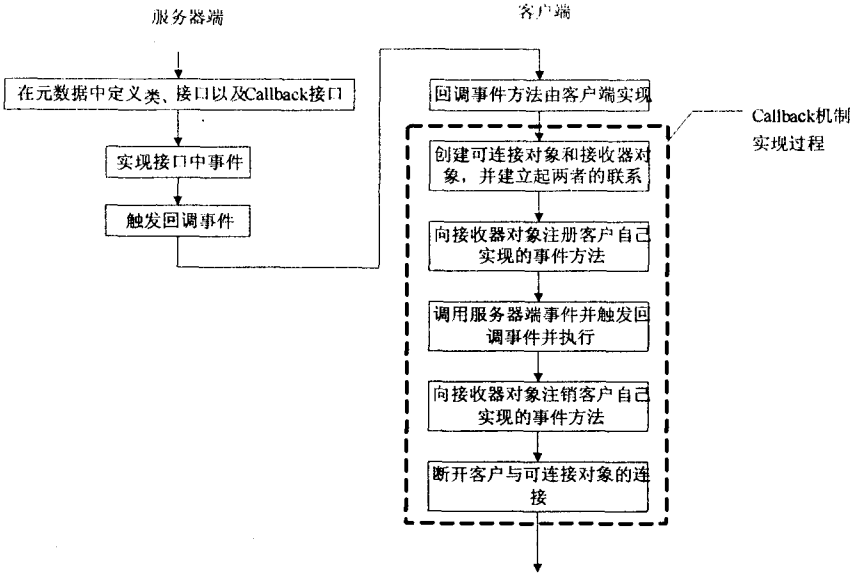

图3　未使用CAR智能指针的Callback机制实现

从图中可以看出,Callback机制实施过程比较复杂,用户需自行创建可连接对象以及接收器,并在接收器中注册对应的回调事件,回调事件触发后,用户还需要向接收器对象注销对应的回调事件,并断开与可连接对象的连接。这一系列过程本身比较复杂,不易于用户的使用。由此,提出采用CAR智能指针为用户封装Callback机制的细节,方便用户的使用。

2.3　CAR智能指针实现回调(Callback)机制

通过CAR智能指针简化用户实现Callback机制的过程如图4所示。

对比图3和图4可以看出,CAR构件技术提供智能指针对客户端进行简化,以方便用户使用CAR构件。

CAR的智能指针向用户屏蔽了对构件回调事件的处理,用户无需再去考虑繁琐复杂的初始化及注册过程,简化了编程的步骤,降低编程难度,同时CAR的智能指针确保客户能正确地控制构件的生命周期,提高了用户应用程序的安全性,降低了资源泄漏的可能;通过智能指针能够检查接口的类型安全性,消除潜在的错误。其实施主要通过创建可连接对象和接收器对象,并建立起两者的联系向接收器对象注册客户自己实现的事件方法AddEventCallbackHandler()以及向接收器对象注销客户自己实现的事件方法,及断开客户与可连接对象的连接事件方法RemoveEventCallbackHandler()来实现。

2.4　CAR智能指针的实现实例

CAR智能指针是一种类智能指针,是对构件类的封装,构件类指的是一个构件中定义的类。在下文中提到的智能指针一律指类智能指针。

在CAR中,类智能指针表现为类,这个类有若干个成员变量,每个成员变量用来指向对象的一个接口,成员变量的数目等于CAR对象实现的接口个数,成员变量和构件对象实现的接口一一对应。通过类智能指针,可以调用构件对象实现的所有接口方法[4]。

类智能指针的表示形式:CxxxRef,其中Cxxx表示类名。一个构件中定义的类分别对应一个相应的类智能指针。

(1) 避免用户忘记释放使用EzCreateObject/new

创建的类对象[5];

(2) 如果该构件类有回调接口,可以直接用智能指针注册回调事件处理函数[5];

(3) 使用智能指针可以方便地访问该构件类的所有接口方法。不需要用 QueryInterface/Query 查询接口[5]。

```
class CSmartpointerSampleRef
{
public: //类 CSmartpointerSamepleRef 的方法声明
  ECODE SampleEvent();
  ECODE AddSampleCallbackEventHandler(ECODE(*)(POBJECT));
  ECODE RemoveSampleCallbackEventHandler(ECODE(*)(POBJECT));
public://运算符重载
  operator IObject*(){return m_pObj;}
  operator ISampleEvent*(){return m_pISampleEvent;}
  operator ICallbackSink*(){return m_sink;}
  IObject** operator &(){return &m_pObj;}
  CSmartpointerSampleRef& operator=(IObject* pObj);
public://智能指针特定的方法
  BOOL ObjIsValid();
  ECODE ObjReferTo(POBJECT pObj);
  ECODE ObjInstantiate(PDOMAININFO pDomainInfo=CTX_SAME_DOMAIN);
public://构造函数和析构函数
  CSmartpointerSampleRef():
    m_pObj(NULL),
    m_pISampleEvent(NULL){}
  ~CSmartpointerSampleRef(){this->ObjDispose();}
private://定义私有变量
  IObject* m_pObj;
  ISampleEvent* m_pISampleEvent;
  CallbackSinkRef m_sink;
private://防止特定的方法被调用
  CSmartpointerSampleRef(const CSmartpointerSampleRef& ref)
  {}
  CSmartpointerSampleRef& operator=(const CSmartpointerSampleRef& ref){return *this;}
};
INLINE ECODE CSmartpointerSampleRef::SampleEvent()
{return m_pISampleEvent->SampleEvent();}
INLINE ECODE CSmartpointerSampleRef::AddSampleCallbackEventHandler(
  ECODE(*fn)(POBJECT))//对注册回调事件进行封装
{EzMultiQI mq={&IID_IObject,NULL,NOERROR};
  ec=EzCreateObjectEx(CLSID_CSmartpointerSample,pDomainInfo,1,&mq);
  if(FAILED(ec))return ec;
  ec=this->ObjReferTo(mq.pObject);
mq.pObject->Release();
  ec=pObj->QueryInterface(IID_ICSmartpointerSample,(POBJECT*)&m_pICSmartpointerSample);
  if(FAILED(ec))goto ErrorExit;
ec=EzCreateObject(CLSID_CSmartpointerSampleSink,CTX_SAME_DOMAIN,IID_ICallbackSink,(POBJECT*)&m_sink);
  if(FAILED(ec))goto ErrorExit;
  ec=m_sink.Connect(pObj);
  if(FAILED(ec))goto ErrorExit;
  return this->AddSampleCallbackEventHandler(NULL,(PVOID)fn);
ErrorExit:
  this->ObjDispose();
  return ec;
}
INLINE ECODE CSmartpointerSampleRef::RemoveSampleCallbackEventHandler(
  ECODE(*fn)(POBJECT))//对注销回调事件进行封装
{if(m_pISampleEvent){
    m_pISampleEvent->Release();
    m_pISampleEvent=NULL;}
  if(m_pObj){
    m_pObj->Release();
    m_pObj=NULL;}
  if(m_sink.ObjIsValid()){
    m_sink.Disconnect();
    m_sink.ObjDispose();}
return this->RemoveSampleCallbackEventHandler(NULL,(PVOID)fn);}
INLINE CSmartpointerSampleRef&CSmartpointerSampleRef::operator=(IObject* pObj)
{
  this->ObjReferTo(pObj);
  return *this;
}
//智能指针类的 ObjIsValid 方法实现,判断智能指针是否有效,此方法为内部方法
INLINE BOOL CSmartpointerSampleRef::ObjIsValid()
{
  if(m_pObj){
    if(TRUE && m_pISampleEvent)return TRUE;
    return NOERROR==this->ObjReferTo(m_pObj);}
  return FALSE;
}
//智能指针类 ObjReferTo 方法实现,初始化智能指针类的接口函数,此方法为内部方法
INLINE ECODE CSmartpointerSampleRef::ObjReferTo(POBJE-
```

(下转第 16 页)

Statement 接口提供了 3 种执行 SQL 语句的方法：executeQuery，executeUpdate 和 execute。使用哪一个方法由 SQL 语句所需产生的内容决定。方法 executeQuery 用于产生单个结果集的语句，例如 SELECT 语句。方法 executeUpdate 用于执行 INSERT，UPDATE 或 DELETE 语句以及 SQL DDL(数据定义语言)语句，例如 CREATE TABLE 和 DROP TABLE。INSERT，UPDATE 或 DELETE 语句的效果是修改表中零行或多行中的一列或多列。executeUpdate 的返回值是一个整数，指示受影响的行数(即更新计数)。对于 CREATE TABLE 或 DROP TABLE 等不操作行的语句，executeUpdate 的返回值总为零。方法 execute 用于执行返回多个结果集、多个更新计数或二者组合的语句。执行语句的所有方法都将关闭所调用的 Statement 对象的当前打开结果集(如果存在)。这意味着在重新执行 Statement 对象之前，需要完成对当前 ResultSet 对象的处理。应该注意，继承了 Statement 接口中所有方法的 PreparedStatement 接口都有自己的 executeQuery，executeUpdate 和 execute 方法。Statement 对象本身不包含 SQL 语句，因而必须给 Statement. execute 方法提供 SQL 语句作为参数。PreparedStatement 对象并不将 SQL 语句作为参数提供给这些方法，因为它们已经包含预编译 SQL 语句。CallableStatement 对象继承这些方法的 PreparedStatement 形式。

(4)处理对数据库的查询结果。

对 authorResults 对象进行处理后，才能将查询结果显示给用户。authorResults 对象包括一个由查询语句返回的一个表，这个表中包含所有的查询结果。对 authorResults 对象的处理必须逐行进行，而对每一行中的各个列，可以按任何顺序进行处理。ResultSet 类的一套 get 方法(这些 get 方法可以访问当前行中的不同列)提供了对这些行中数据的访问，并可将结果集中的 SQL 数据类型转换为 Java 数据类型。

综上所述，即可实现利用 JDBC－ODBC 桥在 Java 中进行数据库的查询。

3 结 语

讨论了利用 Java 语言的 JDBC API 和 JDBC－ODBC 桥完成数据库的 SQL 查询的方法。在此基础上可以构造更为复杂的查询，以满足用户的不同需求。

参考文献：

[1] 李 诚，王 兵. Java 2 简明教程[M]. 北京：清华大学出版社，2004.

[2] 王克宏，张炳文. Java 语言 SQL 接口——JDBC 编程技术[M]. 北京：清华大学出版社，2001：234－262.

[3] 刘 欣. 基于 Servlet 和 JDBC 的 Web 数据库访问方案[J]. 山东电子，2003(1)：18－21.

[4] 刘 巍，唐学兵. 利用 Java 的多线程技术实现数据库的访问[J]. 计算机应用，2002(12)：121－123.

[5] Java2 SDK. Standard edition documentation version 1.3.1 [M]. US：SUN Microsystems，Inc，2003.

[6] Oram，Andy. Database programming with JDBC & Java paperback book[M]. [s.l.]：O'Reilly & Associates，Inc，2000.

(上接第 12 页)

```
CT pObj)
{
  ECODE ec;
  if(NULL = = m_pObj){
    m_pObj = pObj;
    pObj - >AddRef();
  }
  else if(pObj ! = m_pObj){
    this - >ObjDispose();
    m_pObj = pObj;
    pObj - >AddRef();
  }
}
```

3 结 语

用户要使用智能指针，只需采用宏定义 # define SMARTCLASS 将相关智能指针对象引入代码。在实现回调机制时，只需要调用智能指针相关方法完成简单的注册工作就可以使用回调事件，在使用完毕后也只需调用智能指针的相关方法注销该回调事件即可。使用 CAR 智能指针可大大简化 Callback 机制的实现复杂度。

参考文献：

[1] Koretide. Elastos2.0 Manual[M/OL]. 2007. http://www.koretide.com.cn/download/download.php? id=2.

[2] Pan A. COM's Principle and COM's Application[M]. Beijing：The Tsinghua Press，1999.

[3] Rogerson D. Inside COM：Microsoft's Component Object Model[M]. [s.l.]：Microsoft Press.1999.

[4] Eckel B. Thinking in C + + (Second Edition)[M]. [s.l.]：Prentice Hall，2002.

[5] Koretide. Website[EB/OL]. 2007. http://www.koretide.com.cn.