基于CAR构件编程的自动生成Singleton模式的实现

张曼夕,陈 榕,张金焕
(同济大学 基础软件工程中心,上海 200092)

摘 要:设计模式是一种对设计过程中经常遇到的一些问题的总结及其通用的解决方法,使人们可更加简单方便地复用成功的设计和体系结构。Singleton模式属于设计模式中的一种方法。文中通过基于CAR构件编程的方法,设计了一种基于CAR构件编程中实现自动生成Singleton模式的方法,并通过具体实例来证明此方法的可行性与有效性。用户只需要在类设计时指定类属性为Singleton,就可以自动生成Singleton模式的相关实现代码,从而降低了用户编程的复杂度。

关键词:Singleton;CAR;构件
中图分类号:TP311.5 **文献标识码**:A **文章编号**:1673-629X(2008)11-0022-04

Build Automatically Singleton Pattern Based on CAR Component Program

ZHANG Man-xi, CHEN Rong, ZHANG Jin-huan
(System Software Engineering Centre of Tongji University, Shanghai 200092, China)

Abstract: Design pattern which is made people easier to reuse successful design and architecture is a summary on the problems encountered in the design process and its common solution. Singleton pattern belongs to it. Gives a method based on CAR component program, especially introduces a method for building automatic singleton pattern based on CAR component program. The class attribute which is named "singleton" by users is needed when program this class, the relative realized code of singleton mode can be automatically created. And the complexity of programming by users is reduced.

Key words: Singleton; CAR; component

0 引 言

设计模式是指对以上设计过程中经常遇到的一些问题的总结及其通用的解决方法,设计模式使人们可以更加简单方便地复用成功的设计和体系结构。Singleton模式属于设计模式中的一种方法,它的目的是保证一个类仅有一个实例,并提供一个能够访问它的全局访问点。基于CAR构件编程可自动生成Singleton模式,生成相关代码,从而降低用户编程的复杂性。

1 构件编程

1.1 构件技术

构件技术是在面向对象技术的基础上发展起来的。面向构件技术对一组类的组合进行封装,并代表完成一个或多个功能的特定服务,同时为用户提供多个接口。整个构件隐藏了具体的实现,只用接口提供服务。这样,在不同层次上,构件均可以将底层多个逻辑组合成高层次上的粒度更大的新构件。构件之间通过约定的接口进行数据交换和信息传递,构件的位置是相互透明的,可以在同一个用户进程空间,也可以在不同的用户进程空间,甚至在不同的机器上,而且不同的构件可以用不同的语言编写,只要它们符合事先约定的构件规范[1]。

1.2 CAR构件技术

CAR(Component Assembly Runtime)构件技术是一种面向构件编程的编程模型,能够动态加载构件,最大程度支持软件的二进制复用,其兼容微软的COM[2,3]。在一种嵌入式操作系统上,CAR构件[4]技术由CAR语言(构件描述语言,描述构件的元数据信息)、CAR编译器、自动代码生成工具以及CAR构件基础库支持。CAR构件技术体现了网络编程时代的特性,编程界面简单。

CAR技术[5]采用C\C++语言编写构件,所以生成的构件直接是以目标平台的二进制代码运行。相

收稿日期:2008-02-27
基金项目:国家863"软件重大专项"项目(2001AA113400)
作者简介:张曼夕(1982-),女,天津人,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,研究方向为网络操作系统、构件技术。

比 Java、.NET 技术的中间代码和虚拟机机制,它在速度上占有明显的优势,更适合系统级构件的编写以及嵌入式系统中的应用。如果能找到一种在 CAR 构件编程中自动生成 Singleton 模式实现的方法,将大大降低用户编程的复杂程度。

2 Singleton 模式

Singleton 模式属于设计模式中的一种方法,它的目的是保证一个类仅有一个实例,并提供一个能够访问它的全局访问点。这种模式在实际开发中的应用也非常多,例如在嵌入式图形系统应用中,由于内存资源比较宝贵,很多模块都必须保证在整个过程中只能有一个实例。在实际编程设计中,如果要运用该模式,用户可以设计一个 Singleton 的基类,然后在其子类实例中进行各种扩展来配置应用,也可以直接设计一个不允许继承的 Singleton 类,里面包含了相关功能。无论采取哪种方法,用户都必须自行实现 Singleton 模式的相关代码。

考虑一个银行帐户管理系统,其中有上百万个客户帐号,要访问每一个客户的帐号,需要对客户的权限进行验证。在这样的系统里,可以设计一个 Singleton 类叫做 AccountManager,用来管理系统中的客户帐户,此类的实例在整个系统中只有一个,如果用 C# 实现的话,在多线程环境下,可以采用以下方法:

```
public sealed class AccountManager
{
  这个类的其他功能实现代码……
  以下是 Singleton 模式的实现代码:
static AccountManager instance_ =null;
static readonly object lock_ = new Object();
private AccountManager()
    { }
public static AccountManager Instance
  { get
    { lock (lock_)
      { if (instance_ = = null)
        { instance_ = new AccountManager (); }
          return instance_; }}}}
```

以上是 Singleton 模式在 C# 中的基本实现。AccountManager 类被设计成不可继承,当其静态成员变量 instance_ 为空时才在堆中进行类的初始化,其他情况下直接返回该类的实例 instance_ 的值,这样就保证了系统内始终只有一个该类的实例,整个过程通过加锁保证多线程下的一致性。可以看到,不同的用户和不同的语言对于 Singleton 模式的实现可能不相同,但是都必须自己进行 Singleton 类的编写,这样用户在实现类的功能的同时,还必须花精力去实现 Singleton 模式的相关代码,这给用户的编程带来一定的复杂度。

3 基于 CAR 构件编程的自动生成 Singleton 模式的实现

在基于 CAR 构件编程中实现 Singleton 模式,产生 Singleton 的实现代码并非像上述 Java 或者 .NET 通过继承一个 Singleton 基类来获得相应属性,而是直接声明为全局变量并进行初始化,同时对其进行多线程同步管理,保证此全局变量只有一个实例,当编译相应 .car文件后就可自动生成实现此类 Singleton,大大降低了用户编程复杂程度。

在基于 CAR 构件编程中实现 Singleton 模式的方法,当用户在 .car 文件中指定一个类为 Singleton 属性并编译此 .car 文件后,CAR 编译器会自动生成相关的处理代码,其处理过程如下:首先定义几个相关的全局变量,除了有对应此对象的全局指针之外,还有一个状态值代表此对象当前的状态,以及一个全局锁进行同步。状态值分为三种:_SingletonObjState_Uninitialize,代表此对象尚未初始化;_SingletonObjState_Initializing,代表此对象正在进行初始化;_Singleton ObjState_Initialized,代表此对象之前已初始化。然后对状态值加锁并对全局状态值进行检测判断,对对象进行初始化工作,同时相应地设置全局状态值为正在初始化;当全局状态值为正在初始化时,表明已经有其他的线程进行对象的初始化工作,此时该线程将会进入等待状态,并且通过全局状态值不断检测正在分配对象的线程的状况;当全局状态值为已初始化时,获得全局对象变量的值;最后返回对象的值。用户在判断以上三种状态值并进行相应的处理之前,首先会执行加锁的动作,因为状态值为全局变量,对其进行设置必须要加锁以保证多线程下不会出现读写冲突。然后用户会根据当前不同的状态值进入下面相应的程序分支,如图 1 所示。

当对象状态为_SingletonObjState_Uninitialize,表明对象还没有被别的线程初始化过,此时将进行初始化的工作,其过程为首先将对象的全局状态值设为_SingletonObjState_Initializing,表明当前对象正在进行初始化,由于进入分支之前已经加锁,所以当此状态值设置完毕后可以进行解锁,让其他线程也可以读写此状态值。接着在堆中分配一个新的 CAR 构件对象,并将此对象的引用计数加 1(这个过程中如果分配失败,就会重新将对象状态设为_SingletonObjState_Unini-

tialize 并返回,让其他线程进行初始化),然后调用该对象的_InitializeComponent()函数进行一些设置(这个过程如果失败,同样也会将对象状态设为_SingletonObjState_Uninitialize,并返回),最后在返回前才将对象的状态值设为_SingletonObjState_Initialized,表明此构件对象已经被当前线程初始化完毕。当对象状态为_SingletonObjState_Initializing,则表明其他线程正在对此构件对象进行初始化,由于进入分支前已经加锁,此时可以解锁并切换至其他线程继续进行处理,然后重新加锁判断全局状态值,如果状态值为_SingletonObjState_Uninitialize,则表示其他线程可能初始化失败,此时解锁并返回过程开始处由本线程重新进行初始化;如果状态值仍然是_SingletonObjState_Initializing,则表明其他线程还没有把对象初始化完毕,此时应该返回到本分支开始处重新开始解锁以及唤醒其他线程再加锁判断状态值这些步骤,直到检测到对象状态值为_SingletonObjState_Initialized 为止,此时退出此分支进入_SingletonObjState_Initialized 分支。

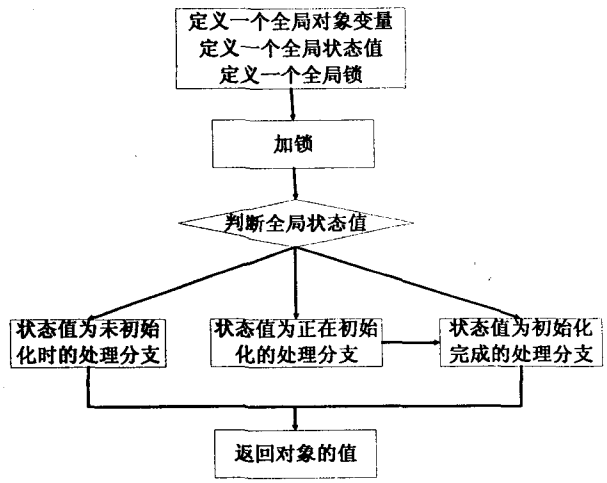

图 1 流程图

4 实例及结果测试

仍以前述银行帐户管理系统作为例子,对于 AccountManager 类,可以在.car 文件中这样指定:

```
module{
    interface IAccountManager { enter(); }
    [singleton]//指定该类为 singleton 类
    class CAccountManager
      { interface IAccountManager;}}
```

CAR 语言中的 module 描述表示一个构件模块,让带有 Singleton 属性的 CAccountManager 类实现了 IAccountManager 接口,为了简单起见,IAccountManager 接口中只包含了一个 enter()方法。

使用 CAR 的编译器以及 CAR 构件的自动代码生成工具对该.car 文件进行处理就会生成构件的实现,CAR 工具会生成 CAccountManager 接口函数的空的实现,而用户只需关心自己定义的接口函数的实现,只要填入自己的代码就可以了。下面是生成的 cpp 文件和 h 文件:

CAccountManager.cpp:

```
#include <stdio.h>
#include "_CAccountManager.cpp"
ECODE CAccountManager::enter()
{ m_cCount++;
printf("CAccountManager enter %d times. \n", m_cCount);
return NOERROR; }
```

其中,CAccountManager.cpp 是 CAR 工具生成的有关构件对象底层实现的代码。可以看到,在 enter 方法中,设立了一个成员变量 m_cCount 做为计数,当调用该方法时就将它的值打印出来,通过它可以判断该类是否在全局中只有一个实例。

CAccountManager.h:

```
#ifndef _CACCOUNTMANAGER_H_
#define _CACCOUNTMANAGER_H_
#include "_CAccountManager.h"
class CAccountManager : public_CAccountManager
{ public:
      CARAPI enter();
      CAccountManager() :m_cCount(0) {}
      ~CAccountManager()
      { puts("CAccountManager dtor."); }
  private:
      int m_cCount; };
```

可以看到成员变量 m_cCount 在构造函数初始化时会设置成 0。

以下是客户端调用此构件方法的一个过程:

```
#include <stdio.h>
#define SMARTCLASS
$ using AccountManager.dll;
//引用 AccountManager.dll 构件
int main()
{ ECODE ec;
    CAccountManagerRef s1,s2;
    ec=s1.ObjInstantiate();
    //初始化两个 AccountManager 类的实例
    if (FAILED(ec)) goto ErrorExit;
    ec = s2.ObjInstantiate();
    if (FAILED(ec)) goto ErrorExit;
//分别调用两个 AccountManager 类的 enter 方法
    for (int n = 0; n < 3; n++) {
          s1.enter();
```

```
        s2.enter();
        }
    return 0;
ErrorExit:
        printf("Error,ec= %x\n", ec);
        return 1;
}
```

以下是运行的结果:

```
CAccountManager enter 1 times.
CAccountManager enter 2 times.
CAccountManager enter 3 times.
CAccountManager enter 4 times.
CAccountManager enter 5 times.
CAccountManager enter 6 times.
CAccountManager dtor.
```

可以看到,虽然调用的是不同的对象的 enter 方法,但是最终调用的都是同一方法,而且最后调用该类析构函数时只调用了一次。

比较一下当没有为 AccountManager 对象指定 singleton 属性时的运行结果:

```
CAccountManager enter 1 times.
CAccountManager enter 1 times.
CAccountManager enter 1 times.
CAccountManager enter 2 times.
CAccountManager enter 1 times.
CAccountManager enter 2 times.
CAccountManager enter 2 times.
CAccountManager enter 3 times.
CAccountManager enter 2 times.
CAccountManager enter 3 times.
CAccountManager enter 3 times.
CAccountManager dtor.
CAccountManager enter 3 times.
CAccountManager dtor.
```

可以看到,没有指定 Singleton 属性时是分别调用了两个不同对象实例的 enter 方法,并且最后析构时调用了两次。这个结果是符合预期的。

同时也可以看到,在具体的 CAccountManager 类的实现中,并没有包含任何 Singleton 模式的管理代码,用户只需要指定 CAccountManager 类为 Singleton 属性并重新编译后,相同的实现代码立即有了不同的运行结果。用户通过这种方式,可以很方便地用 Singleton 模式去解决其他工程问题。

5　结束语

文中基于 CAR 构件编程,设计了一种 Singleton 模式,通过编译 car 文件后就可以自动生成类的 Singleton 实现。用户不必再关心如何去实现 Singleton 模式本身,只需要把精力放在如何实现该类的其他基本功能上即可。这一设计不仅为在 CAR 构件技术中应用 Singleton 模式自动生成代码提出了方法和手段,同时也降低了用户编程的复杂程度,更利于编程。

参考文献:

[1] 陈　榕,刘艺平.基于构件、中间件的因特网操作系统及跨操作系统的构件、中间件运行平台[R].北京:[出版者不详],2003.

[2] Rogerson D. COM 技术内幕(Inside COM)[M].杨秀章译.北京:清华大学出版社,1999.

[3] Box, Don. Essential COM[M]. Menlo Park, CA: Addison-Wesley Publishing Company, 1998.

[4] 上海科泰世纪有限公司. Elastos CAR 构件与编程技术[EB/OL].2006-09. http://www.kortide.com.cn/.

[5] 上海科泰世纪有限公司.和欣资料大全[EB/OL].2006-09. http://www.kortide.com.cn/

(上接第 21 页)

失的连接,对于管理和维护网络都是非常必要的。由于交换机产品种类众多,具体功能各异,往往缺乏统一便捷的手段来进行这方面的工作。文中提出了一种基于 SNMP 和 ARP 的交换机端口 IP 探测方案,经过在多个平台上的使用验证,能快速准确地获取结果,并具有较强的通用性。方案已经在空中交通管制系统中使用,使得网络监控管理和维护的工作更为简便,取得了令人满意的效果。

参考文献:

[1] 李明江. SNMP 简单网络管理协议[M].北京:电子工业出版社,2007.

[2] 周奕利,杨红雨,覃树建,在 Tru64Unix 上使用 SNMP++获取交换机的流量信息[J].中国民航飞行学院学报,2006,17(2):55-57.

[3] 王　芬,赵梗明. 基于 SNMPv3 网络管理系统的研究和应用[J].计算机技术与发展,2006,16(4):199-202.

[4] Wright G R, Stevens W R. TCP/IP Illustrated, Volume 2: The Implementation[M]. USA: Addison Wesley, 1995.

[5] 陈　然,杜晓黎. 基于统一接口的机群中交换机监控系统的设计[J].计算机工程,2005,31(16):225-227.

[6] Stevens W R, Fenner B, Rudoff A M. Unix 网络编程(第一卷)[M].第 3 版.杨继张,译.北京:清华大学出版社,2006.