构件类别的扩展与应用

谢文彬[1] 陈榕[2]
（1、清华大学深圳研究生院软件工程中心 深圳 518055；2、清华大学信息技术研究院操作系统与中间件技术研究中心 北京 100084）

[摘要] 从构件类别概念的来由和作用开始，本文分析了 COM 技术中对构件类别支持的不足，并对此提出了新的扩展和实现。最后介绍构件类别技术在基于构件编程中的几种应用。

[关键词] 构件类别 COM CAR 构件编程

Extension and Application of Component Category

XIE Wen-bin[1] CHEN Rong[2]
(Software Engineering Center, ShenZhen Graduate Institute of Tsinghua University, ShenZhen 518055, China)

[Abstract] Starting with the origin of Component category in component technique, we consider it an important concept, and the support for it in COM(Component Object Model) is not sufficient. This article introduces some new extensions on component category and the corresponding implement. Finally some applications of the component category is also introduced.

[Key word] Component Category COM CAR

1．构件编程遇到的问题和 category 概念的出现

面向构件对象的编程使应用程序容易于扩展，其原因之一是程序总是通过定义好的接口访问构件。随后有新的、实现了同一接口的、功能更强的构件对象出现后，只需要简单的替换构件对象就可以实现应用程序的动态升级，而无需改动原有的程序。

随着新的构件也不断开发出来，构件数量的增加使对它们的管理和使用也变得困难。基本的 COM 激活方式要求调用者知道精确的类名（即 CLSID），通过 CLSID 调用 CoCreateInstance()创建类的实例。而对于同一个接口，可能有许多构件都提供了相似的实现。通常用户在程序中只是希望能够使用某个接口提供的功能，而不关心具体是哪一个构件完成相应功能；而且有些构件在使用中还需要运行环境提供某些支持才能正常工作。如果从构件库中挨个列举构件对象的信息进行查找－－如 Windows 下就需要搜索注册表中，将是非常繁琐和低效率的。

为了对构件进行分类和使用方便，构件类别（Component Category）的概念出现了。构件类别是一个或多个接口的集合，通常，一个类别中的所有构件都将实现同一组接类别，也可能需要好几个构件类别的服务。

COM 对构件类别的支持，只有基于注册表的记录读写：

HKEY_CLASSES_ROOT\Component Categories\子键下记录了所有的构件类别信息；

口。一般来说，类别（category）的定义是有一定的逻辑和语义的，并不简单的是几个接口的集合，更多的体现接口集合所体现的功能和逻辑上的特性。把相关的一类构件对象，划分到对应的构件类别中，在面向构件的应用编程中可以使程序拥有很好的可配置性和扩展性。

2. COM 对 category 提供的支持和实现

在 COM 中，一个构件类别是一组逻辑相关的 COM 构件，它们共享同一个类别 ID（category ID，即 CATID，是一个 GUID），CATID 作为构件的属性被保存在注册表中。每个构件可能有两个类别属性：“Implemented Categories”和 “Required Categories”。前者标识一个构件是哪些构件类别的成员，即实现了哪些构件类别中的接口，可以提供哪些服务；后者表示一个构件正常运行需要运行环境满足的构件类别条件，比如有的构件在运行时需要访问数据库，就需要运行环境能提供访问数据库的类别接口。一个构件可能同时属于好几个构件

本文的研究工作得到国家高技术研究发展计划（863 计划）项目的支持（项目编号 2001AA113400。）

HKEY_CLASSES_ROOT\CLSID\子键下记录每个构件的信息，构件的类别属性也记录在其中。

为了方便的把构件的类别信息添加到注册表中或是从注册表中读出，COM 提供了 Component Category Manager（一个系统实现的构件），包含了两个接口： ICatRegister 和 ICatInformation：

前者用于构件类别信息的注册，提供了三组方法：

RegisterCategories() ：

注册新的构件类别

RegisterClassImplCategories()

注册构件实现了某些类别的信息

RegisterClassReqCategories()

注册构件运行时需要的类别信息

以及对应三个的 unregister(取消注册)方法；

后者用于从注册表中读出类别相关信息，有如下方法：

EnumCategories :

列举所有系统上注册的类别；

GetCategoryDesc:

获取类别的描述字符信息

EnumClassesOfCategoryes :

枚举属于某一类别的类；

IsClassOfCategories :

判断类是否实现或需要某一类别；

EnumImplCategoriesOfClass :

列举实现了某一类别的类；

EnumReqCtegoriesOfClass :

列举需求某一类别的类；

各个接口函数的详细描述可参见 MSDN。COM 中对构件类别的支持仅有这些关于注册表的操作，可以声明新的构件类别，声明某个构件实现了或需要哪些构件类别。但是构件类别的记录中并没有关于它包含哪些接口的任何信息，也不对一个构件是否真正支持某些构件类别中的接口提供保证。

3．和欣对构件类别的概念进行了扩展，并提供了相应的支持

3.1 和欣操作系统与 CAR 构件技术简介

和欣操作系统是科泰世纪公司开发的具有完全自主产权的 32 位嵌入式操作系统，其特点是完全面向构件技术，它提供了一套完整的系统服务构件及系统 API，为在各种嵌入式设备的硬件平台上运行 CAR 二进制构件提供了统一的编程环境。CAR(Component Assembly Run-Time)技术是和欣操作系统的核心技术。

CAR 技术总结面向对象编程、面向构件编程技术的发展历史和经验，能很好地支持面向以 Web Service（WEB 服务）为代表的下一代网络应用软件开发。它很大程度上借鉴了微软的 COM 技术，保持了和 COM 的兼容性，同时对 COM 进行了重要的扩展。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。同时，CAR 删除了 COM 中过时的约定，禁止用户定义 COM 的非自描述接口；完备了构件及其接口的自描述功能，实现了对 COM 的扩展；对 COM 的用户界面进行了简化包装。从上面的定义中，可以说 CAR 是微软 COM 的一个子集，同时又对微软的 COM 进行了扩展，在和欣 SDK 工具的支持下，使构件编程技术很容易被 C/C++程序员理解并掌握。

3.2 CAR 技术中对构件类别的扩展和支持

COM 规范声称，把虚接口抽象出来，就是实现了二进制的多态。但实际情况并不完全如此：构件客户端在使用构件时，还是要指定构件服务器的 CLSID 来创建构件对象，指定 CLSID 实际上就是指定了构件的实现。因此，把接口抽象出来只是实现了构件方法调用的多态性，并未实现构件创建的多态性。而一个构件的使用总是要经过创建、调用、消亡这三个过程的，只有实现了构件对象创建的多态性，才算得上是完整的实现了构件使用的多态。构件类别就是实现构件创建多态的关键。

对比构件编程和 C++的面向对象编程，前者主要是基于接口的编程，而后者主要是基于类对象的编程。面向对象的编程通过创建基类及继承机制，可以很好的对实际问题进行抽象描述，由上而下的建立清晰的层次关系，并在此基础上通过基类指针指向不同的子类而实现多态。接口反映了构件对象的功能特性，这在一定程度上和类对象是很相似的（实际上 COM 接口在 C++中的描述正是纯虚基类）。而 COM 虽然可以实现对接口的继承，但是没有类别继承的概念，在 category 的概念中，可能好几个接口合在一起表征一个逻辑整体，但是并无法实现对它的继承，因此在一个稍微复杂的系统中常出现某一构件拥有一长串的接口的情况，这在设计和维护上都是很不方便的。

CAR 构件定义语言中，明确的引入了构件类别的概念，定义了 category 关键字，对构件类别的定义和使用都很方便。构件类别的使用方法同 C++中的抽象基类的用法是类似的。一个抽象基类是一组其派生类所必须实现的函数的集合，所以可以说此派生类是相应抽象基类的一个特定的实现。与此类似，一个构件类别是属于此类别的构件所必

须实现的接口集合。属于某构件类别的构件是该构件类别的特定实现。

3.3 使用举例

以打印机驱动构件的实现为例，使用构件类别实现多态方式创建构件对象的步骤如下：

步骤一：定义构件类别

使用 CAR 构件描述语言定义类别：

```
[uuid(02aff0b1-a887-4da7-bf2e-626af6165a56) ]
    category  CatPrinter {
    interface  IPrinter;
    }
```

例子中的 CatPrinter 包含了一个接口：Iprinter，实际上，一个类别中允许定义一个到多个接口。

步骤二：定义构件类

如果某个构件类要属于 CatPrinter 这个类别，那么它必须继承 CatPrinter，如：HP LaserJet 6L 打印驱动构件在 CDL 中的定义方式可以如下例所示：

```
    [
        uuid( ….),
        driver
    ]
    class CHPLaserJet6L : CatPrinter {
       interface ILaserJet6L;
    }
```

类 CHPLasterJet6L 的定义本身包含了 ILaserJet6L 接口，同时由于它继承了 CatPrinter 这个类别，它还包含了 CatPrinter 中的 IPrinter 接口。

此外，在 class 前的 driver 属性声明说明了 CHPLaserJet6L 是一个驱动程序。它还必须实现别外一个隐含的系统接口：IDriver，因为驱动程序本身也是一个类别，系统中所有以 driver 属性定义的构件类都属于驱动程序类别：CatDriver，IDriver 接口正是在 CatDriver 中包含的。

步骤三：实现构件类

使用 CAR 编译器编译定义驱动构件类别及构件类的.CAR 文件，会产生构件的源程序框架。框架中包含了构件类所有接口的方法，构件实现者只要根据构件的功能需要填写这些方法的实现代码即可 s。

以例子中 Cprinter 为例，就必须实现 IPrinter、ILaserJet6L 及 IDriver 三个接口中的所有方法。

步骤四：编译构件程序并自动注册构件类别

代码实现完成后，使用编译程序对构件程序进行编译生成构件 DLL 文件。同时会自动把 CAR 文件中定义的构件类别及构件的类别信息注册到系统中。

注：重复步骤二到步骤四，就可以定义并实现多个属于 CatPrinter 类别的打印驱动构件类。

步骤五：编写构件客户（应用程序）程序

构件程序编译生成后，就可以编写构件客户程序来使用构件提供的功能了。构件客户程序可以通过两种方式来创建驱动对象，一是采用常规的 COM 创建对象的方式：通过指定具体驱动构件的 CLSID，如：CLSID_CHPLaserJet6L；二是使用类别标识（CATID），如 CATID_CatPrinter。

要想使构件对象的创建具有多态性，必须使用第二种方式，在 CAR 技术中，可以使用类别的智能指针来创建驱动对象，如：通过类别 CatPrinter 创建打印驱动对象的代码如下：

```
    #import <printer.dll>
    ……
    CatPrinterRef  catPrinterRef;
    hr = catPrinterRef.Instantiate();
    if (FAILED(hr)) {
       ……
    }
    ……
```

步骤六：运行构件客户程序，创建和使用驱动对象

在应用程序中，通过指定构件类别标识（CATID）来创建驱动对象，系统将取该类别的缺省（default）CLSID 来创建构件对象，并返回用户查询的接口；

构件类别中的缺省 CLSID 的来源有三种：

编译构件程序时，编译工具会自动对构件类别及其所属的构件类进行注册，最后一个注册到构件类别中的 CLSID 为该构件类别的缺省 CLSID。

可以调用系统提供的工具注册构件类别中的缺省类标识。

在没有注册文件的情况下，就使用定义构件类别的那个 DLL 文件。如果该文件中有一个以上的构件类属于该类别，将使用第一个属于该类别的构件类作为缺省类，否则创建对象过程将失败。

从上面的例子可以看出，应用 category 和部分类似 driver 的关键字，可以方便的声明构件的类别，增强了源文件的可读性，整个开发流程也十分清晰和简单。增加新的驱动程序时，应用程序也不需要作任何的变动。这只是一个简单的例子，在较复杂的系统设计中，还可以采用构件类别之间的继承，使结构设计更为清晰。

4．Category 在基于构件的程序开发中的应用

在驱动程序中的应用就是构件类别的一个很典型和有效的应用。类似的，我们还可以举出在移动设备中使用不同的通讯协议的例子，移动设备可能采用不同的网络接入方式和使用不同的网络协议，如蓝牙、GPRS、CDMA 等，在应用程序中却可以使用一致的协议接口，一样的调用 send()、recv()等完成通讯，只是在创建通讯协议模块时由系统根据定制规则找到合适的构件。前面这些可以算是软件配置方面的应用，即程序在启动时每次都需要载入这些类别的构件，对这些构件类别进行配置，选择某一类别的不同构件，可以使应用程序很好的适应不同的运行环境，或者让用户根据自己的需求很方便的进行定制。对于这类应用，可以加上一个简单的配置程序，主要功能就是设定构件类别对应的缺省构件选项，通过 GDI 界面，列出构件类别下的所有构件信息，让管理员或用户进行选择和设定。

除了配置的情况，有些应用需要在软件运行时动态选择某一类别的构件。如媒体播放器的编码解码器的例子，需要在运行时根据要播放的媒体文件格式或编码方式选择适用的编码解码器。这种情况下一种常用的方法是通过 CATID 获得该类别下的所有构件的 CLSID，然后逐个查询构件是否支持当前的应用。这是一种很低效的办法。首先，如果是在 Windows 下，获得 CATID 对应的 CLSID 列表需要查询注册表，我们知道注册表是一个保存了大量系统和应用程序信息的数据库，它并没有保存某一类别下所有构件的记录，需要通过关联查找，实践证明查找过程需要花费不少时间；再者，通过构件的接口方法进行查询的话需要挨个把列表的构件实例化，这也是很费时的。这种情况下可以采用一种代理构件的方法，将 CATID 下的缺省构件设定为一个代理构件——该构件保存本类别的构件信息，它提供接口让应用程序传入需求信息（即定义好的构件属性，如编码解码器的例子该属性可定义为对应的 FourCC 代码），然后直接找到并返回对应的构件。同时代理构件也提供让新构件进行“注册”的接口。

Wrapper 技术在基于构件类别的程序开发中是一项很有用的技术。首先，是软件的复用方面，在许多技术比较成熟的领域，数据格式或是应用技术等都有一定的标准，体现在应用的接口都是大致相同的(实现可能各不相同)，比如驱动程序，一般都会提供相似的 Open()，Read()，Write()，Close()等方法。这种情况下，当我们获得一些别人完成的针对某方面应用的构件时，可能它们引出接口并不完全相同。这种情况下，可以根据应用对这些类似的接口进行总结，定义一个新的类别，然后定义一些新的构件继承该构件类别，即都有相同的接口。对新构件的实现，通过构件的包容(Contain)或聚合(Aggregation)等复用方法将原有的那些构件包装起来，功能性的部分由它们实现。新的构件属于同一类别，拥有相同的接口，可以用在基于构件类别的编程中了。

最后来看一个较复杂的可实现运行时动态构件替换，基于构件类别的程序设计模型[4]：

1.需求分析和框架设计。

首先设计应用程序的框架，将那些需要灵活配置，很可能发生需求变动的功能性部分设计封装到构件中，先关注于将各个功能部分连结在一起的整体设计。这也是面向构件的程序设计的一个特点，主程序更多的是起到一个将功能构件联系在一起的胶水作用。之后的功能添加或改动将基本不影响主程序。

2．各功能构件的实现。

定义一个用于统一配置的构件类别：根据需要设计一些用于属性设置和传入初始运行参数的接口，作为基类别；

定义对应不同功能性的的多个类别（均继承基类别）；

各具体的构件继承于对应类别，即包含相应类别中的接口。实现中可以利用 wrapper 和构件复用等技术；

在这一部分中，设计良好的接口是最为重要的。

3．整体合成与提供配置功能。

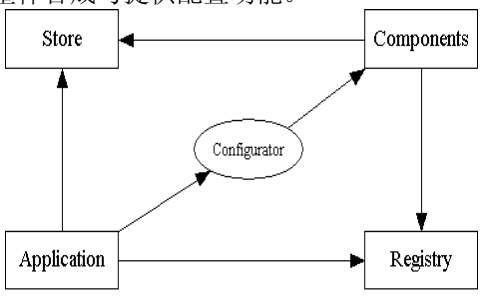

图1

如图 1 所示，Registry 保存了构件类别及其下属构件的所有信息。

假设 Application 作为一个 Web Server 运行，提供某项服务时需要构件类别 A、B、C。在 Application 保持运行的同时，可以用 Configurator 程序选择相应类别中的三个构件，并完成创建和设定初始参数，然后保存

在 Store 对象中，然后 Application 在每次需要提供服务从 Store 对象获取三个构件。在这一过程中，配置类别接口供 configuratou 程序使用；功能性类别接口供 Application 接口使用。

这一设计模式使应用程序具有很大的灵活性，首先可以在运行时动态替换构件；其次新的构件也能够在开发完成后很容易的集成到系统中，并且不需要重新启动程序就可以使用，这在某些应用中是很有用的。

5．总结

本文提到的构件类别是构件编程中的一个重要概念。面向构件编程并不复杂，在程序设计中将需要灵活变动或配置的部分放入构件中实现，并利用构件类别的技术将可以充分发挥构件编程的优势，使程序获得很好的可扩展性、可配置性和适应性。

参 考 文 献

1． COM 本质论 Don Box 著 潘爱民 译 中国电力出版社

2． 和欣资料大全 www.koretide.com.cn

3． 《Writing extensible applications》 By Len Holgate www.codeproject.com

4． 《PluggableComponent －－A pattern for Interactive System Configuration》 By Markus Volter www.voelter.de