异步消息机制原理挖掘与应用范畴分析①

刘鹏远

(湖北经济学院 信息管理学院, 武汉 430205)
(华中科技大学 计算机科学与技术学院, 武汉 430074)

摘 要: 消息机制是面向对象系统和面向对象语言的基本特性, 中间件的异步消息通信方式在现今异构、分布式的网络环境中异步通信得到了广泛应用. 结合 GUI 框架对异步消息通信原理做了剖析, 对其核心框架回调技术做了原理挖掘并不依赖语言、框架给出了实现, 最后给出了异步通信的应用范畴分析, 指出即使在单宿主机内的对象间通信, 采用异步消息通信可改善性能及帮助解决某些基本和特殊问题.

关键词: 消息机制; 回调; 迭代器崩溃; 循环依赖

Digging Principle of Asynchronous Messaging Mechanism and Analysis of Applying Field

LIU Peng-Yuan

(School of Information Management, Hubei University of Economics, Wuhan 430205, China)
(School of Computer Science & Technology of HUST, Wuhan 430074, China)

Abstract: Messaging mechanism is a basic attribute of OO system & OO language. Asynchronous messaging mechanism using middleware is widely spread among heterogeneous and distributive network environment. Combined with GUI framework, a powerful study of multi-procedure asynchronous messaging mechanism was made. Then this paper explored the principle and realized the core callback technology without any framework or developing language. At last it fully analyzed the field of asynchronous messaging application, pointed out that even in single host inner-communication, asynchronous messaging can improve performance better and help to solve some basic and special questions.

Key words: messaging mechanism; callback; iterator collapse; circular dependency

消息机制是面向对象语言和面向对象系统最重要的特性之一. 在面向对象范畴内, 消息是一个对象在自身不能闭合提供所需功能时向外发出的请求, 也就是请求另一个或一群对象提供自身所需的服务, 也称为广义消息. 熟知的各种 GUI 系统都是由消息驱动的, 而且操作系统本身的许多功能, 以及各种简单或复杂的框架系统通常也要依靠消息和消息队列来完成各类主要操作. 这里所说的消息往往和某些特定的系统调用(如 WIN32 中的消息处理函数)和系统功能(如操作系统实现的消息队列)相关, 所承载的大多是系统一级的信息, 如键盘、鼠标灯硬件消息, 通知、刷新等软件消息等. 常将系统一级的消息称为“事件(Event)”, 将支持类似功能的系统称为“事件驱动系统”, 也称这类消息是狭义消息, 以区别于广义消息.有关异步消息通信机制的研究大多集结于分布式开放性异构软件系统间的异步通信: 如专注于应用服务器与消息中间件通信, 主要讨论应用系统和商用中间件间的应用整合[1-3]; 或关注于异步消息常用的中间件如 CORBA 的持续改进[4], 甚少涉及自实现中间件系统和平台级研发. 在硬件开发和实时性要求高的场合, 单宿主机内的进程间通信也有着异步消息的应用范畴, 这方面涉及更底层的消息通信机制原理, 而与中间件以及网络通信无关, 国内外甚少归纳和研究. 本文旨在通过分析异步消息机制原理, 挖掘其核心实现并展开对其应用范畴的进一步分析, 试图解决某些特殊和基本的问题.

① 基金项目:国家自然科学基金(60973085/F020502)
收稿时间:2013-12-09;收到修改稿时间:2014-01-13

1　同异步消息范畴

在UML顺序图或协作图中，一个对象可以直接向另一个对象发送一个特定的消息，这类消息实际分别对应于目的类(接收消息对象所隶属的类)的某个职责. 因此, UM L 中的消息最终会映射成为目的类的方法或函数，这时消息(Message)是对象发出的服务请求，包括请求服务的对象名、服务名、传入的消息参数以及返回的结果类型. 在面向对象分析阶段，通常用消息来指代类/对象间的一次动态联系，在面向对象设计阶段，消息名称被转换为目的类的方法和函数.

根据发出请求和响应的时间关系，消息可分为同步和异步两类.

① 同步消息(Synchronous Message)

对象 A 调用对象 B 的 getData()方法，就是对象 A 向对象 B 发送 getData 消息，方法传递的参数就是消息参数. 这是我们最常见的同步消息. 同步意即发送消息方(Client 调用方)和消息接收方(Server 提供服务方)处于同一串行控制流上.

② 异步消息(Asynchronous Message)

如果一个对象发送消息后，不等消息返回就继续自己的活动，这种消息称之为异步消息. 类似与中断处理：发送消息方发起中断请求后不需等待，继续后继流程处理. 当中断服务完毕时，系统提醒发送方回来接收处理结果. 异步消息的支持需要一个第三方，一般称之为应用框架，将在第 2 节中进行分析.

上述范畴明确指出：同步消息可简单地认为调用对方的一个服务函数，因此同步消息也被称为过程调用(Procedure Call). 可视为是同一进程内部的一条串行控制流；异步消息则有着应用系统框架作为中介存在，适用于不同进程间的通信.

2　多进程间异步消息通信机制

明晰消息的分类后，还要明确何时应用同步消息、何时必须考虑应用异步消息进行通信. 显然，同步消息是简单的函数调用通信，通信两者间并无中介而是直接通信. 明确的一点是，当同步消息的处理函数处理时间过长但该调用要求要有较好的实时性时，应用同步消息不合适——这意味着一个较长的不可打断的等待. 2.1 节将结合常用的 Windows GUI 系统，描述揭示异步消息通信的一般过程. 2.2 节则试图摆脱任何框架系统，利用面向对象语言本身特质模拟回调这一异步通信的核心. 2.3 节则试图摆脱语言限制，给出核心回调的原理性实现.

2.1 Windows GUI 消息机制

在 Windows 操作系统和各种支持图形用户界面 GUI 的操作系统中，不同的应用程序间通过向窗口对象发送消息来互相通信，接收到消息的对象负责对消息的响应处理. 窗口对象是附属于不同的应用程序的部件，例如从 A 应用程序窗口切换到 B 应用程序窗口，可看成是 A 窗口向 B 窗口发送了消息，那么 A 究竟是如何向 B 程序发出消息的呢？

Windows 操作系统下进程间是相互隔离的，由于 Windows 对进程地址空间的保护，进程间不能直接调用对方的功能函数，它们必须间接通过操作系统提供的进程间通信机制来相互联系，也就是说，请求服务的发送消息端和响应请求的接受消息端是相互隔离的，通信过程是一个异步过程. Windows 系统中两个拥有窗口的进程 A/B 间，A 向 B 请求服务的消息过程如顺序图 1、图 2 所示:

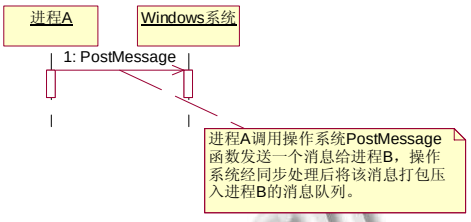

图 1　A 向操作系统发消息

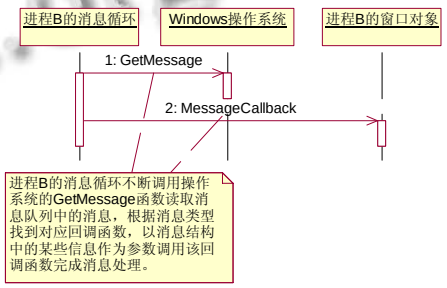

图 2　B 取操作系统中保护的消息队列

进程 A 与 B 间被消息循环(消息泵)隔离，这实际是微软提出的消息队列(Message Queue，MQ)通信方式.

图 2 中的回调函数即响应相应请求的消息处理函数，它实际上不是直接由请求方调用的，而是在某个异步时刻间接由操作系统或框架层调用的函数[5]. 一

般化来形象理解, 能被系统 “自动调用”的函数被称之为回调函数, 从通信方向上看, 它是从底层调用上层服务, 有别于一般意义上的上层调用底层服务.

本节给出的 GUI 例子具有通用性, 不论采用何种消息中间件或操作系统 GUI, 进程间的异步通信都采用通过第三方中介的形式进行, 中间有着一个巧妙地被称为回调的调用方向转向的过程, 这显著不同于同步消息的直接过程调用方式.

2.2 支持继承多态的面向对象系统下回调的实现方法

2.1 节描述了 GUI 系统下进程间通信的依赖系统框架的异步通信方式, 本节将不依赖框架进一步分析异步消息通信原理.

联系面向对象思想的基本原则中的依赖倒置原则[6], 它也具有类似回调的特点, 即调用转向:

1) 基类定义默认实现----------框架层定义一个粗糙的默认实现, 适用于框架系统.

2) 不同派生类均覆盖默认实现, 提供了定制化的不同业务逻辑----------应用层提供个性化的定制解决方案, 适用于购买框架进行二次具体适配开发的组织.

3) 客户仅依赖于基类类型的调用----------客户层客户不太可能在源代码级进行系统的操作使用(有研发能力的客户单位也较多采用联合开发系统方式介入), 也就是说, 客户对系统的调用体是以 UI 形式固化的, 尽管如此, 实际发生的调用取决于传入的对象类型——很可能是某具体派生类的对象.

将(1)-(3)视为不同层的对象, 比如可将基类视为框架层中的代码, 派生类视为应用层的代码, 客户端调用属于测试的用户层. 那么, 客户在 UI 上输入的实际参数的变化会导致好像形式上对底层框架的调用实际转向了对某合适派生类对象的业务逻辑调用, 方向上看起来就是对底层框架的调用转向了对实际上层业务逻辑层的调用.

因此, 核心回调的原理在于实现一种调用转向. 借助 OOP 语言的该动态多态机制可实现核心回调. 即可对消息处理函数也做类似处理:

1) 在框架层中定义稳定的基类接口(粗糙的、一般化的、或空的默认实现).

2) 在应用层由具体应用具体定义新的派生类各种消息处理函数.

3) 客户层运行系统时框架层会自动地导向去调用定制化的派生类消息处理函数, 实际覆盖了基类业务逻辑.

分析图 3 给出的通用链式栈的多态例子, 显然客户能自定义任意类型的数据来使用该通用链式栈, 这里自定义的数据类型就是客户变化的需求, 需要开发者设计类图予以满足该灵活易变的需求. 为此构造一个抽象接口类型 BaseType, 并约定客户希望出入栈的任意自定义类型数据必须实现这个接口. 当有新类型要求使用栈时, 实际就是该接口的一种接口实现, 由客户自行扩展接口实现.

对服务端需提供的栈来说, 定义对 BaseType 类型适用的出入栈操作即可. 由于 Client 通过稳定的接口函数 push/pop 来操作栈, 因此该栈模块无须修改可通用.

当发生新的数据类型要出入栈时, 客户定义新的类型类继承实现 BaseType 接口, 然后就可以直接使用栈压入和弹出该类型数据.

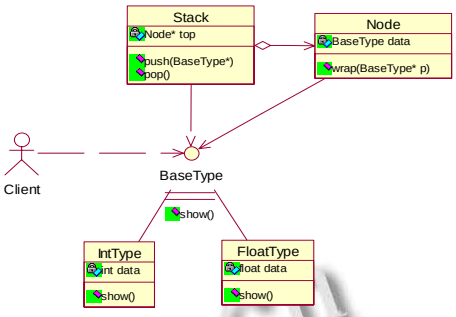

图 3 通用链式栈的多态

图 3 中, 居于下方的 IntType, FloatType 等自定义类型是由 Client 依据 BaseType 接口自行扩展的类, 居于上方的 Stack 和 Node 是服务端封装的类. 当有新的类型需要使用这个栈时, 用户自行扩展新类.

本节联系面向对象的动态多态机制, 发现其与异步通信的核心回调有着类似的调用转向特质, 借助 OOP 动态多态给出了回调的实现原理, 从而摆脱了框架层的依赖阐述了回调的核心实质.

2.3 无多态机制支持下的回调实现方法

在相当多的嵌入式和强调实时性的开发环境中, OOP 语言反而不及汇编/C 语言普及, 这主要是由于实时性的要求使得只有中级、甚至低级语言方能满足要求. 因此, 如何在该类原始开发环境中得到类似 2.2 节的异步通信回调效果值得研究.

在不支持多态特性如 C 语言这种面向过程语言的

开发环境中, 仍然可将客户端和业务函数提供端、甚至框架层视为不同层的对象, 这样的视野能更好地将开发系统分层, 并有利于分工协作. 非 OO 语言没有继承多态支持, 此时的回调可使用函数指针来实现:

1) 在框架层中定义回调函数的接口(函数声明, 需要保持相对稳定), 它具有 OO 中基类虚函数通用和默认实现的特点.

2) 在应用层实现特定函数并将函数指针传递给框架层(函数指针建立指向), 这种特定实现具有派生类覆盖改写新业务逻辑的特点.

3) 当实际执行相关功能时, 框架层就直接使用函数指针调用, 实际上回调了应用层实现的特定函数.

下面给出一个示例, 完成的功能非常简单: 比较传入的相同类型的数据大小. 为简略展示回调的实现原理, 这里假定只能输入可比较的数据类型参数(整型/浮点/字符等), 并分框架层声明、框架层定义、应用层声明、应用层定义, 以及测试体驱动五层分别展开分析:

```
//1 框架层声明 frame.h
#ifndef _FRAME
#define _FRAME
#include <stdio.h>
void *c,*d;
int (*pcompare)(void *a, void *b);//函数指针
//建立回调用的钩子函数
void mapping(void *a,void *b,int (*pf)(void*a,void*b));
void process();//执行业务函数
#endif
```

本部分定义了框架层稳定的接口, 给出了具有较为通用版本的比较两个数大小函数指针版本 pcompare 函数原型. 注意到这是一个具有通用性质的比较函数, 为此使用了通用指针类型, 而具体的类型指定以及 pcompare 函数的实现, 由应用层负责实现.

mapping 则是一种钩子函数(Hook)[6], 也就是映射的意思. 即通过一种赋值转换, 建立框架层和应用层之间的联系.

```
//2 框架层定义 frame.c
#include "frame.h"
//映射函数, 建立框架层与应用层的钩子
void mapping(void *a ,void *b, int (*pf)(void*a,void*b))
{  c=a; d=b; pcompare=pf;  }
//业务函数
void process()
{  switch(pfunc(c,d))
     { case 0: printf("the two is equal!\n");
       case 1: printf("the fronter is larger than the behinder!\n");
       case -1:printf("the fronter is little than the behinder!\n");
     }  }
```

本部分包含框架层两个重要函数的实现: process 是框架层的业务默认实现, 实际是直接调用函数指针指向的函数(由应用层实现). Mapping 则是建立回调联系的函数, 它将外部传入的回调函数指针传递给框架层函数指针 pcompare 保存, 将业务参数传递给框架层内部变量保存.

```
//3 应用层声明 app.h
int compare(void *a,void *b);
```

本部分包括应用层待实现的比较函数声明, 该声明应与框架层定义的函数指针类型保持一致.

```
//4 应用层定义 app.c
#include "frame.h"
#include "app.h"
int compare(void *a,void *b)
{  int *pa=(int *)a, *pb=(int *)b;
     if (*pa==*pb)      return 0;
       else if (*pa>*pb)    return 1;
   else return -1;  }
```

本部分是应用层特定函数的具体实现, 这里的 compare 是 int 版本的特定实现. 只需客户端给出不同的具体实现, 框架层保持稳定, 能以不变应万变.

```
//5 客户端, 即测试体驱动
#include "app.h"
void main()
{  int a=3,b=4;
  mapping(&a,&b,compare);
    process();  }
```

main 函数首先通过框架层提供的 mapping 函数建立将应用层实现的回调函数与框架层勾连起来, 以传递实际函数地址和需要比较的业务数据; 然后在某个时刻调用框架层的执行业务函数 process 函数, 实际执行的是应用层实现的 int 版本的 compare 函数.

本节不依赖 OOP 的动态多态, 仅使用 C 语言这类基础性中级语言以分层代码形式给出了回调实现的完

整例子. 从而从根本上揭示了回调的原理.

3 单宿主机单进程内异步消息通信应用范畴

发起服务的请求者(Client)和响应请求的服务提供者(Server)常不位于同一宿主机(Host)内时, 这种多进程间的通信多见于分布式网络环境下的通信, 甚至是异构系统间的通信. 一般采用中间件通信, 如使用著名的 CORBA 通信框架. 而在同一宿主机内, 不同对象间的异步通信常采用的是类似第 2 节举例的 Windows GUI 框架通过框架层回调处理来进行异步消息通信的方法, 比如将一visio文档图复制粘贴到word文档. 除此外, 同一进程内部的不同对象间也能使用异步消息机制来处理一些棘手的特殊问题(3.2 和 3.3 节将进行分析).

也就是说, 异步消息机制既天然自然地适用于多进程通信, 也能应用于单进程内部来解决一些特殊需求或问题. 当面临这些特殊需求时, 使用同步消息机制处理会出现一些问题. 下面分节列出异步消息通信应用于单宿主机单进程内的应用范畴.

3.1 异步消息用于提高系统的吞吐率和运行效率

容易想到可使用异步消息机制消去无谓的等待: 比如一个消息处理可能要耗费很长时间, 发出同步消息请求的一方一直等待是非常浪费 CPU 时间的事. 于是将请求和响应改为异步的方式, 发送方将消息结构填入消息队列, 接收方轮询队列查找自己的回调函数进行后继处理.

3.2 异步消息用于消除循环依赖造成的无限递归

循环依赖[7]对软件系统灵活性构成很大损害, 且可能具有隐蔽性不易发现. 考虑下面的单进程内部消息请求序列, 对象 A—>对象 B->对象 C->对象 A->..., 结果发生了无限递归, 且后面的请求序列无法得到响应机会. 这种间接递归隐蔽性强, 各对象间并不知道发生了循环递归, 而我们又允许这种消息传递上的间接递归, 认为这是不可预见的合理存在形式. 可以采用采用异步的方法改造上面的流程以消除函数重入.

每个对象发出消息后, 将消息存入先进先出的消息队列然后继续后继事务处理, 接收者采用轮询方式查询消息队列并处理自己的消息, 这种处理机制非常类似数据库事务处理中避免死锁的资源顺序申请分配制. 由于消息总是被发送者按照先后顺序压入到队列中, 而消息接收者也总是按照这样的顺序去检索队列, 能够避免类似互相需要对方服务造成的死锁.

3.3 异步消息用于避免迭代器崩溃

“迭代器崩溃”是一类非常典型的问题[8], 使用异步消息机制能避免“迭代器崩溃”:

假设对象 A 需要遍历一个数组(迭代器), 向数组中的每个对象元素发出一个消息, 而某个接收到消息的对象元素 a 有删除该数组元素的响应, 例如收到的是脱离该数组的 detach 消息. 如果使用同步消息来实现, 那么该迭代器在还没有遍历到a后面的对象元素b时发生崩溃.

解决方法和上面的例子类似, 改造为异步方式. 对象 A 在遍历完数组才发出消息才将它们填入到一个队列, 然后才开放该队列给其它所有对象轮询匹配消息处理函数.

4 小结

本文较为深入地分析了消息机制原理, 从 WIN GUI 入手引入描述了多进程间的异步消息通信过程, 其核心是一种具有调用转向的回调技术, 并借用 OOP 语言的动态多态机制对回调做了原理实现描述, 随后脱离语言框架限制, 采用 C 语言分层代码详述了回调的实现原理. 最后分节指出在单进程内部的不同对象间通信也可以采用异步消息通信用以解决部分特殊和基本问题.

参考文献

1 金花,朱亚涛,靳志强,郑娜.异步通讯机制下应用服务器与消息中间件整合的研究.河北农业大学学报,2009,32(4):112−115.

2 邱岩,王卫兵.消息中间件的集群技术.计算机工程,2003,29(2):107−108.

3 赵艳红,郭银章,白尚旺.基于时间解耦的分布对象中间件异步通信消息转换机制研究.计算机应用与软件,2008,25(10):160−162.

4 张志伟,隋品波,郭长国,吴泉源.分布对象中间件异步消息的研究与实现.计算机学报,2004,27(12):1626−1631.

5 刘鹏远,温珏,桂超,李祥.面向对象UML系统分析建模.北京:清华大学出版社,2013.

6 Flower M.重构:改善既有代码的设计.侯捷译.北京:中国电力出版社,2003.

7 刘鹏远,邓沌华,李祥.不同粒度循环依赖的消去方法.信息通信,2013,(6):9−10.

8 Eckel B. Thinking in C++. Volume1, 2nd ed. Upper Saddle River, NJ: Prentice Hall, 2000.