**POCO C++库学习和分析 -- 通知和事件 （一）**

**1. 信息交流的方法**

        在讨论Poco中事件与通知之前，先来聊一聊信息交流的方法，这样或许有助于理解接下去的讨论。我们都知道数据之间存在关系。在数据库模型里，关系被分为一对一，一对多，多对多。在用计算机去解决数据关系的时候，多对多关系往往被分解成为数个一对多，而一对多的关系最终被分解成为数个一对一关系。  
       如果用关系的观点去看消息流动，消息存在一个或多个发起者，即消息源Source；消息也存在一个或多个接收者，即目标对象Target；同时消息Message本身具有内容，即多种消息。简化成为最终的一对一模型，来描述消息的，那么这个模型里的三个要素就是，Source，Message，Target。  
  
       把这个模型放到C++语言中，Source，Message，Target分别被抽象成为三个类。那么消息传送的方式可以被写成如下两种方式：

**1.1 放置模型于编程语言**

       从**消息源的角度**来考虑问题，整个消息发送的流程是：  
       a) 目标向消息源注册， Source.Register(Target)  
       函数实现大致是这样的

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8664372)

1. Source.Register(Target)
2. {
3. source.vec.add(Target);
4. }

       b) 消息产生  
       Source create a msg  
       c) 消息发送，Soucre.Send(Msg)  
       上面这个函数大致如下:

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8664372)

1. Soucre.Send(Msg)
2. {
3. Target.Receive(Msg);
4. }

       Target.Receive(Msg)的函数实现大致是这样的,

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8664372)

1. Target.Receive(Msg)
2. {
3. **switch**(Msg)
4. {
5. **case** Msg1:
6. doing something;
7. **case** Msg2:
8. doing something;
9. **default**:
10. doing something;
11. }
12. }

       这种方式的调用可以说是最常见能够想到的方法了。最早写C代码的时候，就开始使用了，到C++时代也是。  
  
       换个角度，在C++时代一切都是对象，从**消息的角度**来考虑这个问题呢，于是整个消息发送流程变成为：  
       a) 目标向消息注册， Msg.Register(Target);  
       Msg.Register(Target)函数实现大致是这样的

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8664372)

1. Msg.Register(Target)
3. Msg.vec.add(Target);

       b) 消息产生  
       a msg create by some one source   
       c) 消息发送，Msg.Send(Source)  
       Msg.Send(Source)函数实现大致是这样的,

**[cpp]** [view plaincopy](http://blog.csdn.net/arau_sh/article/details/8664372)

1. Msg.Send(Source)
2. {
3. **switch**(Source)
4. {
5. **case** Source1:
6. doing something;
7. **case** Source2:
8. doing something;
9. **default**:
10. doing something;
11. }
12. }

       上面就是Poco中通知与事件的大致思路。其中通知是站在消息源的角度来考虑问题，而事件是站在消息的角度来考虑问题。插一句话，Poco中的事件和代理来自于C#。也就是说，分析Poco中的事件，其实是在解释C#的代理和事件的实现。

**1.2 放置模型于多线程环境**

       让我们抛开语言吧，把消息传递的过程放到多线程当中去。用多线程干吗？消息的产生就是为了最终的处理，假如消息处理很耗时间怎么办？  
       没办法，兜里没银子。那句话怎么说来着，高富帅猛升硬件，穷挫矮死搞算法。毕竟大家不全是铁道部，是不？于是乎把消息的产生和处理放在两个线程中不是挺好的一个主意吗，这样毫无疑问消息处理的效率得到了提升。这也就是生产者和消费者模式。  
       如果从这个角度考虑的问题的话，那么我们得到了消息传递的另外一种划分。同步处理消息和异步处理消息。  
  
       于是我们可以把消息的传递过程分为下面4种：  
  
  
           
                    通知    |    事件  
                                |  
       同步    (支持)  |  (支持)    
             ———————————  
       异步    (支持)  |  (支持)    
                               |  
  
       下面的章节我们将对Poco中消息和事件一一进行分析。

（版权所有，转载时请注明作者和出处  <http://blog.csdn.net/arau_sh/article/details/8664372>）