



(12)发明专利

(10)授权公告号 CN 106648816 B

(45)授权公告日 2020.03.17

(21)申请号 201611127674.3

(22)申请日 2016.12.09

(65)同一申请的已公布的文献号
申请公布号 CN 106648816 A

(43)申请公布日 2017.05.10

(73)专利权人 武汉斗鱼网络科技有限公司
地址 430000 湖北省武汉市东湖开发区软
件园东路1号软件产业4.1期B1栋11楼

(72)发明人 李从章

(74)专利代理机构 北京众达德权知识产权代理
有限公司 11570

代理人 刘杰

(51)Int.Cl.
G06F 8/41(2018.01)

(56)对比文件

CN 104503918 A,2015.04.08,
CN 104239037 A,2014.12.24,
CN 103019823 A,2013.04.03,
CN 103677844 A,2014.03.26,
US 2013212598 A1,2013.08.15,
CN 103780680 A,2014.05.07,

审查员 庄文龙

权利要求书1页 说明书9页 附图2页

(54)发明名称

多线程处理系统及方法

(57)摘要

本发明公开了一种多线程处理系统及方法，其中的系统包括订阅者接口模块、发布者模块和订阅者模块；其中：所述订阅者接口模块，用于提供订阅者接口，所述订阅者接口被所述订阅者模块继承；所述发布者模块，用于提供注册接口函数从而注册订阅者，建立发布者与订阅者关联，以及用于创建多线程，利用所述多线程监听网络消息或进行逻辑处理，并将网络消息或者逻辑处理结果异步传递给订阅者；所述订阅者模块，用于提供若干个用户界面类，利用所述用户界面类监听所述网络消息或者逻辑处理结果。本发明可提高多线程开发效率。



1. 一种多线程处理系统,其特征在於,包括订阅者接口模块、发布者模块和订阅者模块;其中:

所述订阅者接口模块,用于提供订阅者接口,所述订阅者接口被所述订阅者模块继承;

所述发布者模块,用于提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及用于创建多线程,利用所述多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理结果异步传递给订阅者;

所述订阅者模块,用于提供若干个用户界面类,利用所述用户界面类监听所述网络消息或者逻辑处理结果;

所述发布者模块还用于,在发布消息给订阅者之前,对发布的消息进行异步处理;

其中,所述系统基于MFC开发平台;所述发布者模块进行异步处理具体包括:自定义分派消息,并绑定所述分派消息和分派函数;利用应用程序接口传送函数发送所述分派消息,从而调用在界面线程中执行的分派函数;其中,所述分派函数与所述传送函数均在同一所述界面线程中执行;

或者,所述系统基于QT开发平台;所述发布者模块进行异步处理具体包括:自定义异步信号分派消息,将分派函数设置为槽函数,通过连接函数将所述异步信号分派消息与所述分派函数进行绑定;通过信号发射函数发送所述异步信号分派消息时,调用在界面线程中执行的分派函数;其中,所述发射函数与所述分派函数均在同一所述界面线程中执行。

2. 如权利要求1所述的系统,其特征在於,所述发布者模块还用于继承所述订阅者接口模块提供的所述订阅者接口。

3. 一种多线程处理方法,其特征在於,所述方法包括:

提供订阅者接口,所述订阅者接口被订阅者继承;

提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及创建多线程,利用所述多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理的结果异步传递给订阅者;

提供若干个用户界面类,利用所述用户界面类监听所述网络消息或者逻辑处理结果;

其中,在发布消息给订阅者之前,所述方法还包括:对发布的消息进行异步处理;

所述方法基于MFC开发平台;所述异步处理具体包括:自定义分派消息,并绑定所述分派消息和分派函数;利用应用程序接口传送函数发送所述分派消息,从而调用在界面线程中执行的分派函数;其中,所述分派函数与所述传送函数均在同一所述界面线程中执行;

或者,所述方法基于QT开发平台;所述异步处理具体包括:自定义异步信号分派消息,将分派函数设置为槽函数,通过连接函数将所述异步信号分派消息与所述分派函数进行绑定;通过信号发射函数发送所述异步信号分派消息时,调用在界面线程中执行的分派函数;其中,所述发射函数与所述分派函数均在同一所述界面线程中执行。

4. 如权利要求3所述的方法,其特征在於,所述订阅者接口被发布者继承。

多线程处理系统及方法

技术领域

[0001] 本发明涉及计算机软件技术领域,具体涉及一种多线程处理系统及方法。

背景技术

[0002] 多线程处理是一个技术难点,容易出现错误,而且调试复杂。例如有两个线程,一般主线程会开辟一个单独的线程处理容易阻塞的事务,目前很多的处理办法是通过回调函数(callback)来调用,但是callback函数的处理线程并不是主线程,这样callback函数还是需要考虑多线程。目前多线程处理方案中多个线程之间的交互较为复杂,处理效率低下。

发明内容

[0003] 鉴于上述问题,提出了本发明以便提供一种克服上述问题或者至少部分地解决上述问题的多线程处理系统及方法,可提高多线程处理的效率。

[0004] 依据本发明的一个方面,提供一种多线程处理系统,包括订阅者接口模块、发布者模块和订阅者模块;其中:所述订阅者接口模块,用于提供订阅者接口,所述订阅者接口被所述订阅者模块继承;所述发布者模块,用于提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及用于创建多线程,利用所述多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理结果异步传递给订阅者;所述订阅者模块,用于提供若干个用户界面类,利用所述用户界面类监听所述网络消息或者逻辑处理结果。

[0005] 优选的,所述发布者模块还用于继承所述订阅者接口模块提供的所述订阅者接口。

[0006] 优选的,所述发布者模块还用于,在发布消息给订阅者之前,对发布的消息进行异步处理。

[0007] 优选的,所述系统基于MFC开发平台;所述发布者模块进行异步处理具体包括:自定义分派消息,并绑定所述分派消息和分派函数;利用应用程序接口传送函数发送所述分派消息,从而调用在界面线程中执行的分派函数。

[0008] 优选的,所述系统基于QT开发平台;所述发布者模块进行异步处理具体包括:自定义异步信号分派消息,将分派函数设置为槽函数,通过连接函数将所述异步信号分派消息与所述分派函数进行绑定;通过信号发射函数发送所述异步信号分派消息时,调用在界面线程中执行的分派函数。

[0009] 依据本发明的一个方面,提供一种多线程处理方法,所述方法包括:提供订阅者接口,所述订阅者接口被订阅者继承;提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及创建多线程,利用所述多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理的结果异步传递给订阅者;提供若干个用户界面类,利用所述用户界面类监听所述网络消息或者逻辑处理结果。

[0010] 优选的,所述订阅者接口被发布者继承。

[0011] 优选的,还包括:在发布消息给订阅者之前,对发布的消息进行异步处理。

[0012] 优选的,所述方法基于MFC开发平台;所述异步处理具体包括:自定义分派消息,并绑定所述分派消息和分派函数;利用应用程序接口传送函数发送所述分派消息,从而调用在界面线程中执行的分派函数。

[0013] 优选的,所述方法基于QT开发平台;所述异步处理具体包括:自定义异步信号分派消息,将分派函数设置为槽函数,通过连接函数将所述异步信号分派消息与所述分派函数进行绑定;通过信号发射函数发送所述异步信号分派消息时,调用在界面线程中执行的分派函数。

[0014] 可见,通过本发明提供的多线程处理系统可简化处理复杂度。多线程开发一直是程序开发中的难点,调试也比较特殊,是错误高发地。本发明实施例使用观察者设计模式,将发布者的网络消息异步传递给订阅者,使得所有函数都在用户界面类主线程中执行,由此使得界面开发者不会碰到多线程,从而从多线程的复杂中解脱开来,这样就避免了使用多线程可能会导致的种种如同步等方面的错误,大大的提高了开发的效率以及程序的稳定性。特别的,QT和MFC是PC界面开发最常用的两大平台,本方案优选方案中利用MFC和QT平台本身的特性,使得ui所有的函数都是在ui主线程中执行。

[0015] 上述说明仅是本发明技术方案的概述,为了能够更清楚了解本发明的技术手段,而可依照说明书的内容予以实施,并且为了让本发明的上述和其它目的、特征和优点能够更明显易懂,以下特举本发明的具体实施方式。

附图说明

[0016] 通过阅读下文优选实施方式的详细描述,各种其他的优点和益处对于本领域普通技术人员将变得清楚明了。附图仅用于示出优选实施方式的目的,而并不认为是对本发明的限制。而且在整个附图中,用相同的参考符号表示相同的部件。在附图中:

[0017] 图1示出了根据本发明实施例的多线程处理系统架构示意图;

[0018] 图2示出了根据本发明实施例的多线程处理系统示例示意图;

[0019] 图3示出了根据本发明实施例的多线程处理方法流程图。

具体实施方式

[0020] 下面将参照附图更详细地描述本公开的示例性实施例。虽然附图中显示了本公开的示例性实施例,然而应当理解,可以以各种形式实现本公开而不应被这里阐述的实施例所限制。相反,提供这些实施例是为了能够更透彻地理解本公开,并且能够将本公开的范围完整的传达给本领域的技术人员。

[0021] 本发明针对开发含有多线程例如网络通信的情况,通过开发平台自身的特性,并利用观察者设计模式封装,简化多线程交互,使界面开发者从复杂的多线程的漩涡中解脱开来,界面开发者所有的操作都在自己的界面主线程中。

[0022] 上述的开发平台,可以是指目前较为流行的QT或MFC。下面首先对QT、QT信号和槽、MFC、设计模式、观察者模式、相关接口和函数进行介绍。

[0023] QT是跨平台C++图形用户界面应用程序开发框架,它既可以开发GUI程序,也可用于开发非GUI程序,比如控制台工具和服务器。QT是面向对象的框架,使用特殊的代码生成

扩展(称为元对象编译器(Meta Object Compiler,moc))以及一些宏,易于扩展,允许组件编程。QT很容易扩展,并且允许真正地组件编程。

[0024] 信号和槽机制是QT的核心机制,应用于对象之间的通信,它独立于标准的C/C++语言,因此要正确的处理信号和槽,必须借助一个称为moc(Meta Object Compiler)的QT工具,该工具是一个C++预处理程序,它为高层次的事件处理自动生成所需要的附加代码。信号的声明是在头文件中进行的,QT的signals关键字指出进入了信号声明区,随后即可声明自己的信号。槽是普通的C++成员函数,可以被正常调用,它们唯一的特殊性就是很多信号可以与其相关联。当与其关联的信号被发射时,这个槽就会被调用。槽可以有参数,但槽的参数不能有缺省值。所有从QObject或其子类(例如QWidget)派生的类都能够包含信号和槽。当对象改变其状态时,信号就由该对象发射(emit)出去。当一个信号被发射时,与其相关联的槽将被立刻执行,就象一个正常的函数调用一样。

[0025] MFC(Microsoft Foundation Classes)是微软基础类库的简称,是微软公司实现的一个c++类库,主要封装了大部分的windows API函数,并且包含一个应用程序框架,以减少应用程序开发人员的工作量。其中包含的类包含大量Windows句柄封装类和很多Windows的内建控件和组件的封装类。

[0026] 设计模式(Design pattern)是一套被反复使用、多数人知晓的、经过分类编目的、代码设计经验的总结。使用设计模式是为了可重用代码、让代码更容易被他人理解、保证代码可靠性。设计模式使代码编制真正工程化。

[0027] 观察者模式(又被称为发布(publish)-订阅(Subscribe)模式、模型-视图(View)模式、源-收听者(Listener)模式或从属者模式)是软件设计模式的一种。在此种模式中,一个目标物件管理所有相依于它的观察者物件,并且在它本身的状态改变时主动发出通知。实现观察者模式有很多形式,比较直观的一种是使用一种“注册-通知-撤销注册”的形式。

[0028] PostMessage是Windows API(应用程序接口)中的一个常用函数,该函数将一个消息放入(寄送)到与指定窗口创建的线程相联系消息队列里,不等待线程处理消息就返回,是异步消息模式。消息队列里的消息通过调用GetMessage和PeekMessage取得。

[0029] connect是QT中QObject类的成员函数,该函数用于将信号发送者sender对象中的信号signal与接受者receiver中的member槽函数联系起来。

[0030] emit是Qt的信号发射函数,当一个信号被发射时,与其相关联的槽将被立刻执行。

[0031] 参见图1,为本发明实施例的多线程处理系统架构示意图。

[0032] 该系统包括订阅者接口模块101、发布者模块102和订阅者模块103。

[0033] 订阅者接口模块101,用于提供订阅者接口,该订阅者接口被订阅者模块103继承。

[0034] 发布者模块102,用于提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及用于创建多线程,利用多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理的结果异步传递给订阅者。这里可以理解,发布者模块102也继承订阅者接口模块,表示其本身也是特殊的订阅者。发布者模块102在发布消息给其他订阅者之前,会首先对发布的消息做处理,然后才发布,这里的处理即做异步的处理。

[0035] 订阅者模块103,用于提供若干个用户界面类,利用用户界面类监听发布者的消息,并对收到的消息进行处理。

[0036] 参见图2,为根据本发明实施例的多线程处理系统示例示意图。该示例使用了观察者模式,即“发布(Publish)-订阅(Subscribe)”模式。IMsgListener为订阅类接口(订阅者接口模块101的具体形式),NetOP为发布类(发布者模块102的具体形式),IMsgListener子类通过NetOP::register()函数订阅其关心的消息类型,当对应消息来到时,NetOP会通过IMsgListener::on_msg()函数交给订阅者处理。IMsgListener接口类:信息监听的基类,UI中如果有类需要处理相关网络消息,都需要继承该接口。IMsgListener::on_msg()函数为监听子类中处理接收到的消息。

[0037] NetOP类:本方案的主要类,主要功能含有创建线程监听网络消息或处理复杂逻辑,然后把消息或处理结果反馈给订阅者,并且订阅者对接收到的消息的处理是在ui线程中进行的。注意该类也继承IMsgListener接口类。

[0038] NetOP::register()函数:用于注册ui类订阅者,当对应的消息接收之后会分派给订阅者进行相关处理,订阅者需要继承IMsgListener。

[0039] NetOP::dispatch_msg函数:用于分派消息给订阅者。

[0040] NetOP::thread_proc()函数:用于监听网络消息或处理复杂逻辑线程函数。

[0041] NetOP::work函数:监听网络消息或处理复杂逻辑处理

[0042] NetOP::make_msg函数:构建消息

[0043] UserWin1,UserWin2类(订阅者模块103的具体形式):用户ui类,继承IMsgListener接口类,实现on_msg接口函数处理其订阅的相关消息。

[0044] 下面对图2示例的具体实现过程进行介绍。

[0045] 第一步,创建(New)一个NetOP类对象实例,创建相关ui类,通过NetOP::register()函数向NetOP订阅。这个步骤为准备工作。

[0046] 第二步,NetOP类开启线程处理工作。

[0047] 创建新的线程,在该线程中监听网络消息或进行复杂逻辑处理,并发布消息,该线程函数的伪代码示例如下:

```
NetOP::thread_proc(void* arg)
{
NetOP* _THIS = (NetOP*)arg;
While(true)
{
[0048] if(_THIS ->stop())
break;
if(_THIS ->work() && _THIS ->make_msg(stMsg) )
_THIS->on_msg(stMsg);
}
}
```

[0049] 该线程函数需要传递NetOP类对象的指针,该线程函数一直执行,直到NetOP::stop()返回为真,表示线程处理结束。NetOP::work函数监听网络消息或进行复杂逻辑处理,返回true表示接收到网络消息或有逻辑处理反馈,然后传给构建消息并交给NetOP::on_msg处理,该函数会通过NetOP::dispatch_msg函数分派网络消息。

[0050] 第三步,NetOP对接收到网络消息或复杂逻辑的反馈结果的处理。

[0051] 由前面可知,NetOP在发布消息给其他订阅者之前,需要先在NetOP::on_msg函数中进行处理,该函数对消息进行异步处理,异步处理的结果是NetOP::on_msg函数与NetOP::dispatch_msg的函数的执行是在不同的线程,即NetOP::dispatch_msg分派消息的函数不是thread_proc中,而是在ui线程中执行。

[0052] 该函数的异步处理需要使用到开发平台(例如mfc或qt平台)的特性,。在mfc中,可使用PostMessage,qt中需要使用信号和槽,这两者从前面的介绍中可以知道,都支持异步执行。

[0053] 在mfc中,自定义一个用户自定义消息WM_DISPATCH_MSG,PostMessage函数需要一个窗口句柄,所以需要把一个窗口类继承NetOP,并把消息WM_DISPATCH_MSG与NetOP::dispatch_msg绑定。在NetOP::on_msg函数,可使用PostMessage函数发送自定义windows消息WM_DISPATCH_MSG,也会使得NetOP::dispatch_msg被调用,该方法是使用了windows的消息队列,这里,NetOP::dispatch_msg函数执行也是在ui线程中的。

[0054] 在Qt中,可在NetOP中定义信号signal_dispatch_msg,NetOP::dispatch_msg函数设置为槽函数,通过connect函数把该信号和dispatch_msg槽函数绑定(注意connect参数中必须是异步的ConnectionType,由于使用到信号与槽,NetOP需要继承QObject),在NetOP::on_msg函数使用emit发送signal_dispatch_msg信号,该信号会使得NetOP::dispatch_msg函数被调用,由于该信号是异步的,NetOP::dispatch_msg函数执行已经是在

ui线程中了。

[0055] 第四步,把网络消息分配给相关订阅类对象。

[0056] 前述步骤已经把消息异步传递给了dispatch_msg函数,该函数是在ui线程执行的,这样UserWin1、UserWin2消息订阅类的on_msg()函数完全不需要考虑多线程了。dispatch_msg函数中将根据消息的类型,调用对应的订阅类的on_msg()函数处理网络消息。

[0057] 通过本发明提供的多线程处理系统可简化处理复杂度。多线程开发一直是程序开发中的难点,调试也比较特殊,是错误高发地。本发明实施例使用观察者设计模式,将发布者的网络消息异步传递给订阅者,使得所有函数都在用户界面类主线程中执行,由此使得界面开发者不会碰到多线程,从而从多线程的复杂中解脱开来,这样就避免了使用多线程可能会导致的种种例如同步等方面的错误,大大的提高了开发的效率以及程序的稳定性。特别的,QT和MFC是PC界面开发最常用的两大平台,本方案优选方案中利用MFC和QT平台本身的特性,使得ui所有的函数都是在ui主线程中执行。

[0058] 下面通过检测多个ip连接状况为应用背景,对本发明的一个具体实例进行介绍。

[0059] 假设有100个ip,通过“ping”命令来测试各个ip的连接状况。实现方法是:对每个ip ping 10次,根据ping值确认连接状况(例如ping的时间少于80毫秒,确定成功,如果成功几率在60%以上,确定ip之间的连接状况可接受)。具体实现中,可通过进度条显示进度,状态栏显示成功率。

[0060] 该实例完全可以使用本方案进行处理:这里把复杂的“ping”工作使用NetOP来处理,需要监听进度消息的进度条和显示成功率的状态栏使用UserWin1、UserWin2来代替。根据前面的NetOP::thread_proc的实现,我们只需要实现NetOP::work(),NetOP::make_msg函数即可,示例代码如下:

```
NetOP:: work ()
{
    If(get_one_ip(std::string& strIP)) //获取一个 ip
    {
        Ping_10(strIP.c_str()); // 每个 ip ping10 次 , 并根据规则判断该 ip 连
[0061] 接状况 , ture 表示 ok , false 表示连通不上
        return true;
    }
    Sleep(10);
    return false;
}
```



```
Struct stMsg{  
    Float  fPercentPing;  
[0062]    Float  fPercentSuccess;  
}
```

[0063] NetOP::make_msg(stMsg)函数只需要根据ping的ip数量和成功的数量以及总的ip的数量计算相应百分比填充stMsg消息即可。

[0064] 下面解释现有技术处理和本发明处理的不同。

[0065] 现有技术处理：

```
NetOP::thread_proc(void* arg)  
{  
    NetOP*  _THIS = (NetOP*)arg;  
    While(true)  
    {  
[0066]    if(_THIS ->stop())  
        break;  
        if(_THIS ->work() && _THIS ->make_msg(stMsg) )  
            _THIS-> dispatch_msg (stMsg);  
    }  
}
```

[0067] 现有方案一般在新线程thread_proc中直接调用dispatch_msg()函数进行消息分发,即dispatch_msg()函数直接调用订阅者UserWin1,UserWin2的on_msg函数发送消息,这样dispatch_msg()函数调用订阅者的on_msg的处理是在thread_proc线程中,而不是在ui线程中,默认UserWin1,UserWin2是执行在ui线程中的。由此,就导致UserWin1,UserWin2的实现是跨了线程的(即thread_proc线程和ui线程)

[0068] 本发明处理：

```
NetOP::thread_proc(void* arg)
{
NetOP* _THIS = (NetOP*)arg;
While(true)
{
[0069] if(_THIS ->stop())
break;
if(_THIS ->work() && _THIS ->make_msg(stMsg) )
_THIS->on_msg(stMsg);
}
}
```

[0070] 本方案在thread_proc线程中没有直接调用dispatch_msg()函数进行消息派发,如图2,这里发布者模块NetOP也继承订阅者接口模块,表示其本身也是特殊的订阅者,发布者模块NetOP在发布消息给其他订阅者之前,会首先对发布的消息做处理,然后才发布。这里的处理是异步的处理,即on_msg处理了跨线程的问题,通过MFC的windows信息队列(QT中的异步信号槽)机制,使得dispatch_msg函数的执行已经在ui线程中了,由此,on_msg的处理也是在ui线程中,保证了UserWin1,UserWin2的实现都在ui线程中的,

[0071] 由以上对比可以发现,只有NetOP需要处理多线程的问题,而UserWin1、UserWin2等界面ui类函数的执行始终在ui线程中,即在一个线程中,由此ui编程者就回避了多线程的处理。

[0072] 参见图3,为本发明提供的一种多线程处理方法,该方法包括:

[0073] S301:提供订阅者接口,该订阅者接口被订阅者继承;

[0074] S302:提供注册接口函数从而注册订阅者,建立发布者与订阅者关联,以及创建多线程,利用多线程监听网络消息或进行逻辑处理,并将网络消息或者逻辑处理的结果异步传递给订阅者;

[0075] S303:提供若干个用户界面类,利用用户界面类监听发布者的消息,并对收到的消息进行处理。

[0076] 优选的,订阅者接口被发布者继承。

[0077] 优选的,该方法还包括:在发布消息给订阅者之前,对发布的消息进行异步处理。

[0078] 优选的,所述方法基于MFC开发平台;所述异步处理具体包括:自定义分派消息(WM_DISPATCH_MSG),并绑定所述分派消息和分派函数(NetOP::dispatch_msg);利用应用程序接口传送函数(PostMessage)发送所述分派消息,从而调用在界面线程中执行的分派函数。

[0079] 优选的,所述方法基于QT开发平台;所述异步处理具体包括:自定义异步信号分派消息(signal_dispatch_msg),将分派函数(NetOP::dispatch_msg)设置为槽函数,通过连接函数(connect)将所述异步信号分派消息与所述分派函数进行绑定;通过信号发射函数(emit)发送所述异步信号分派消息时,调用在界面线程中执行的分派函数。

[0080] 在此提供的算法和显示不与任何特定计算机、虚拟系统或者其它设备固有相关。各种通用系统也可以与基于在此的示教一起使用。根据上面的描述,构造这类系统所要求的结构是显而易见的。此外,本发明也不针对任何特定编程语言。应当明白,可以利用各种编程语言实现在此描述的本发明的内容,并且上面对特定语言所做的描述是为了披露本发明的最佳实施方式。

[0081] 本发明的各个部件实施例可以以硬件实现,或者以在一个或者多个处理器上运行的软件模块实现,或者以它们的组合实现。本领域的技术人员应当理解,可以在实践中使用微处理器或者数字信号处理器(DSP)来实现根据本发明实施例的用户变身控制的系统中的一些或者全部部件的一些或者全部功能。本发明还可以实现为用于执行这里所描述的方法的一部分或者全部的设备或者装置程序(例如,计算机程序和计算机程序产品)。这样的实现本发明的程序可以存储在计算机可读介质上,或者可以具有一个或者多个信号的形式。这样的信号可以从因特网网站上下载得到,或者在载体信号上提供,或者以任何其他形式提供。

[0082] 应该注意的是上述实施例对本发明进行说明而不是对本发明进行限制,并且本领域技术人员在不脱离所附权利要求的范围的情况下可设计出替换实施例。在权利要求中,不应将位于括号之间的任何参考符号构造成对权利要求的限制。单词“包含”不排除存在未列在权利要求中的元件或步骤。位于元件之前的单词“一”或“一个”不排除存在多个这样的元件。本发明可以借助于包括有若干不同元件的硬件以及借助于适当编程的计算机来实现。在列举了若干装置的单元权利要求中,这些装置中的若干个可以是通过同一个硬件项来具体体现。单词第一、第二、以及第三等的使用不表示任何顺序。可将这些单词解释为名称。



图1

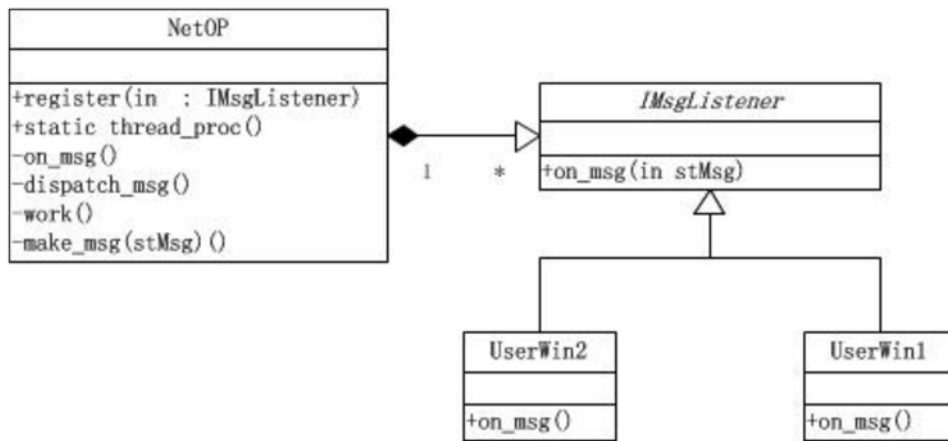


图2

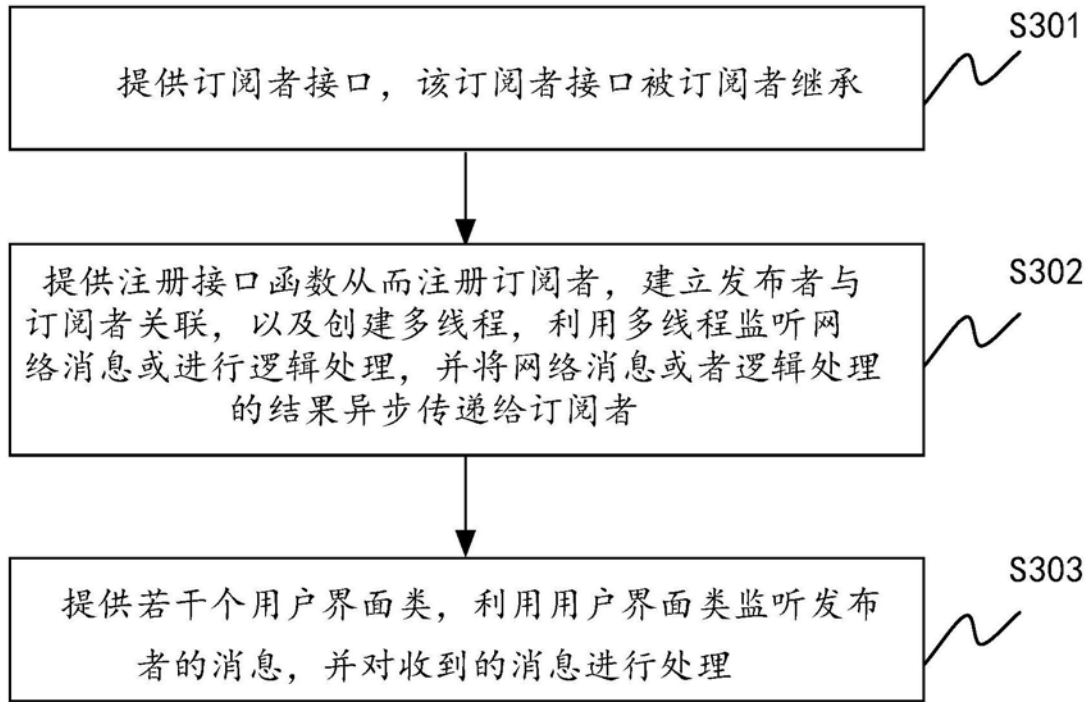


图3