

附录C 消息传递框架与完整的ATM示例

ATM：自动取款机。

1回到第4章，我举了一个使用消息传递框架在线程间发送信息的例子。这里就会使用这个实现来完成ATM功能。下面完整代码就是功能的实现，包括消息传递框架。

清单C.1实现了一个消息队列。其可以将消息以指针(指向基类)的方式存储在列表中；指定消息类型会由基类派生模板进行处理。推送包装类的构造实例，以及存储指向这个实例的指针；弹出实例的时候，将会返回指向其的指针。因为message_base类没有任何成员函数，在访问存储消息之前，弹出线程就需要将指针转为wrapped_message指针。

清单C.1 简单的消息队列

```
1  #include <mutex>
2  #include <condition_variable>
3  #include <queue>
4  #include <memory>
5
6  namespace messaging
7  {
8      struct message_base // 队列项的基础类
9      {
10         virtual ~message_base()
11         {}
12     };
13
14     template<typename Msg>
15     struct wrapped_message: // 每个消息类型都需要特化
16         message_base
17     {
18         Msg contents;
19
20         explicit wrapped_message(Msg const& contents_):
21             contents(contents_)
22         {}
23     };
24
```

```

25  class queue  // 我们的队列
26  {
27      std::mutex m;
28      std::condition_variable c;
29      std::queue<std::shared_ptr<message_base> > q;  // 实际存储指向message_base类
30  public:
31      template<typename T>
32      void push(T const& msg)
33      {
34          std::lock_guard<std::mutex> lk(m);
35          q.push(std::make_shared<wrapped_message<T> >(msg));  // 包装已传递的信息，有
36          c.notify_all();
37      }
38
39      std::shared_ptr<message_base> wait_and_pop()
40      {
41          std::unique_lock<std::mutex> lk(m);
42          c.wait(lk, [&]{return !q.empty();});  // 当队列为空时阻塞
43          auto res=q.front();
44          q.pop();
45          return res;
46      }
47  };
48  }
```

发送通过`sender`类(见清单C.2)实例处理过的消息。只能对已推送到队列中的消息进行包装。对`sender`实例的拷贝，只是拷贝了指向队列的指针，而非队列本身。

清单C.2 sender类

```

1  namespace messaging
2  {
3      class sender
4      {
5          queue*q;  // sender是一个队列指针的包装类
6      public:
7          sender():  // sender无队列(默认构造函数)
8              q(nullptr)
9          {}
10
11         explicit sender(queue*q_):  // 从指向队列的指针进行构造
12             q(q_)
13         {}

```

```
14
15     template<typename Message>
16     void send(Message const& msg)
17     {
18         if(q)
19         {
20             q->push(msg); // 将发送信息推送给队列
21         }
22     }
23 };
24 }
```

接收信息部分有些麻烦。不仅要等待队列中的消息，还要检查消息类型是否与所等待的消息类型匹配，并调用处理函数进行处理。那么就从**receiver**类的实现开始吧。

清单C.3 receiver类

```
1 namespace messaging
2 {
3     class receiver
4     {
5         queue q; // 接受者拥有对应队列
6     public:
7         operator sender() // 允许将类中队列隐式转化为一个sender队列
8         {
9             return sender(&q);
10        }
11        dispatcher wait() // 等待对队列进行调度
12        {
13            return dispatcher(&q);
14        }
15    };
16 }
```

sender只是引用一个消息队列，而**receiver**是拥有一个队列。可以使用隐式转换的方式获取**sender**引用的类。难点在于**wait()**中的调度。这里创建了一个**dispatcher**对象引用**receiver**中的队列。**dispatcher**类实现会在下一个清单中看到；如你所见，任务是在析构函数中完成的。在这个例子中，所要做的工作是对消息进行等待，以及对其进行调度。

清单C.4 dispatcher类

```
1 namespace messaging
```

```

2  {
3      class close_queue  // 用于关闭队列的消息
4      {};
5
6      class dispatcher
7      {
8          queue* q;
9          bool chained;
10
11          dispatcher(dispatcher const&)=delete;  // dispatcher实例不能被拷贝
12          dispatcher& operator=(dispatcher const&)=delete;
13
14          template<
15              typename Dispatcher,
16              typename Msg,
17              typename Func>  // 允许TemplateDispatcher实例访问内部成员
18          friend class TemplateDispatcher;
19
20          void wait_and_dispatch()
21          {
22              for(;;)  // 1 循环，等待调度消息
23              {
24                  auto msg=q->wait_and_pop();
25                  dispatch(msg);
26              }
27          }
28
29          bool dispatch(  // 2 dispatch()会检查close_queue消息，然后抛出
30              std::shared_ptr<message_base> const& msg)
31          {
32              if(dynamic_cast<wrapped_message<close_queue>*>(msg.get()))
33              {
34                  throw close_queue();
35              }
36              return false;
37          }
38      public:
39          dispatcher(dispatcher&& other):  // dispatcher实例可以移动
40              q(other.q),chained(other.chained)
41          {
42              other.chained=true;  // 源不能等待消息
43          }
44
45          explicit dispatcher(queue* q_):
46              q(q_),chained(false)

```

```

47     {}
48
49     template<typename Message,typename Func>
50     TemplateDispatcher<dispatcher,Message,Func>
51     handle(Func&& f)  // 3 使用TemplateDispatcher处理指定类型的消息
52     {
53         return TemplateDispatcher<dispatcher,Message,Func>(
54             q,this,std::forward<Func>(f));
55     }
56
57     ~dispatcher() noexcept(false)  // 4 析构函数可能会抛出异常
58     {
59         if(!chained)
60         {
61             wait_and_dispatch();
62         }
63     }
64 };
65 }

```

从`wait()`返回的`dispatcher`实例将马上被销毁，因为是临时变量，也向前文提到的，析构函数在这里做真正的工作。析构函数调用`wait_and_dispatch()`函数，这个函数中有一个循环①，等待消息的传入(这样才能进行弹出操作)，然后将消息传递给`dispatch()`函数。`dispatch()`函数本身②很简单；会检查小时是否是一个`close_queue`消息，当是`close_queue`消息时，抛出一个异常；如果不是，函数将会返回`false`来表明消息没有被处理。因为会抛出`close_queue`异常，所以析构函数会标示为 `noexcept(false)`；在没有任何标识的情况下，一般情况下析构函数都 `noexcept(true)` ④型，这表示没有任何异常抛出，并且`close_queue`异常将会使程序终止。

虽然，不会经常的去调用`wait()`函数，不过，在大多数时间里，你都希望对一条消息进行处理。这时就需要`handle()`成员函数③的加入。这个函数是一个模板，并且消息类型不可推断，所以你需要指定需要处理的消息类型，并且传入函数(或可调用对象)进行处理，并将队列传入当前`dispatcher`对象的`handle()`函数。这将在清单C.5中展示。这就是为什么，在测试析构函数中的`chained`值前，要等待消息耳朵原因；不仅是避免“移动”类型的对象对消息进行等待，而且允许将等待状态转移到新的`TemplateDispatcher`实例中。

清单C.5 TemplateDispatcher类模板

```

1  namespace messaging
2  {
3      template<typename PreviousDispatcher,typename Msg,typename Func>
4      class TemplateDispatcher
5      {

```

```

6      queue* q;
7      PreviousDispatcher* prev;
8      Func f;
9      bool chained;
10
11     TemplateDispatcher(TemplateDispatcher const&)=delete;
12     TemplateDispatcher& operator=(TemplateDispatcher const&)=delete;
13
14     template<typename Dispatcher,typename OtherMsg,typename OtherFunc>
15     friend class TemplateDispatcher; // 所有特化的TemplateDispatcher类型实例都是友元
16
17     void wait_and_dispatch()
18     {
19         for(;;)
20         {
21             auto msg=q->wait_and_pop();
22             if(dispatch(msg)) // 1 如果消息处理过后, 会跳出循环
23                 break;
24         }
25     }
26
27     bool dispatch(std::shared_ptr<message_base> const& msg)
28     {
29         if(wrapped_message<Msg>* wrapper=
30             dynamic_cast<wrapped_message<Msg>*>(msg.get())) // 2 检查消息类型, 并且
31         {
32             f(wrapper->contents);
33             return true;
34         }
35         else
36         {
37             return prev->dispatch(msg); // 3 链接到之前的调度器上
38         }
39     }
40 public:
41     TemplateDispatcher(TemplateDispatcher&& other):
42         q(other.q),prev(other.prev),f(std::move(other.f)),
43         chained(other.chained)
44     {
45         other.chained=true;
46     }
47     TemplateDispatcher(queue* q_,PreviousDispatcher* prev_,Func&& f_):
48         q(q_),prev(prev_),f(std::forward<Func>(f_)),chained(false)
49     {
50         prev_->chained=true;

```

```

51     }
52
53     template<typename OtherMsg,typename OtherFunc>
54     TemplateDispatcher<TemplateDispatcher,OtherMsg,OtherFunc>
55     handle(OtherFunc&& of)  // 4 可以链接其他处理器
56     {
57         return TemplateDispatcher<
58             TemplateDispatcher,OtherMsg,OtherFunc>(
59             q,this,std::forward<OtherFunc>(of));
60     }
61
62     ~TemplateDispatcher() noexcept(false)  // 5 这个析构函数也是noexcept(false)的
63     {
64         if(!chained)
65         {
66             wait_and_dispatch();
67         }
68     }
69 };
70 }

```

`TemplateDispatcher<>`类模板仿照了`dispatcher`类，二者几乎相同。特别是在析构函数上，都是调用`wait_and_dispatch()`等待处理消息。

在处理消息的过程中，如果不抛出异常，就需要检查一下在循环中①，消息是否已经得到了处理。当成功的处理了一条消息，处理过程就可以停止，这样就可以等待下一组消息的传入了。当获取了一个和指定类型匹配的消息，使用函数调用的方式②，就要好于抛出异常(虽然，处理函数也可能会抛出异常)。如果消息类型不匹配，那么就可以链接前一个调度器③。在第一个实例中，`dispatcher`实例确实作为一个调度器，当在`handle()`④函数中进行链接后，就允许处理多种类型的消息。在链接了之前的`TemplateDispatcher<>`实例后，当消息类型和当前的调度器类型不匹配的时候，调度链会依次的前向寻找类型匹配的调度器。因为任何调度器都可能抛出异常(包括`dispatcher`中对`close_queue`消息进行处理的默认处理器)，析构函数在这里会再次被声明为`noexcept(false)` ⑤。

这种简单的架构允许你想队列推送任何类型的消息，并且调度器有选择的与接收端的消息进行匹配。同样，也允许为了推送消息，将消息队列的引用进行传递的同时，保持接收端的私有性。

为了完成第4章的例子，消息的组成将在清单C.6中给出，各种状态机将在清单C.7,C.8和C.9中给出。最后，驱动代码将在C.10给出。

清单C.6 ATM消息

```
1  struct withdraw
2  {
3      std::string account;
4      unsigned amount;
5      mutable messaging::sender atm_queue;
6
7      withdraw(std::string const& account_,
8              unsigned amount_,
9              messaging::sender atm_queue_):
10         account(account_), amount(amount_),
11         atm_queue(atm_queue_)
12     {}
13 };
14
15 struct withdraw_ok
16 {};
17
18 struct withdraw_denied
19 {};
20
21 struct cancel_withdrawal
22 {
23     std::string account;
24     unsigned amount;
25     cancel_withdrawal(std::string const& account_,
26                      unsigned amount_):
27         account(account_), amount(amount_)
28     {}
29 };
30
31 struct withdrawal_processed
32 {
33     std::string account;
34     unsigned amount;
35     withdrawal_processed(std::string const& account_,
36                        unsigned amount_):
37         account(account_), amount(amount_)
38     {}
39 };
40
41 struct card_inserted
42 {
43     std::string account;
44     explicit card_inserted(std::string const& account_):
45         account(account_)
```



```
46     {}
47 };
48
49 struct digit_pressed
50 {
51     char digit;
52     explicit digit_pressed(char digit_):
53         digit(digit_)
54     {}
55 };
56
57 struct clear_last_pressed
58 {};
59
60 struct eject_card
61 {};
62
63 struct withdraw_pressed
64 {
65     unsigned amount;
66     explicit withdraw_pressed(unsigned amount_):
67         amount(amount_)
68     {}
69 };
70
71 struct cancel_pressed
72 {};
73
74 struct issue_money
75 {
76     unsigned amount;
77     issue_money(unsigned amount_):
78         amount(amount_)
79     {}
80 };
81
82 struct verify_pin
83 {
84     std::string account;
85     std::string pin;
86     mutable messaging::sender atm_queue;
87
88     verify_pin(std::string const& account_, std::string const& pin_,
89               messaging::sender atm_queue_):
90         account(account_), pin(pin_), atm_queue(atm_queue_)
```

```
91     {}
92 };
93
94 struct pin_verified
95 {};
96
97 struct pin_incorrect
98 {};
99
100 struct display_enter_pin
101 {};
102
103 struct display_enter_card
104 {};
105
106 struct display_insufficient_funds
107 {};
108
109 struct display_withdrawal_cancelled
110 {};
111
112 struct display_pin_incorrect_message
113 {};
114
115 struct display_withdrawal_options
116 {};
117
118 struct get_balance
119 {
120     std::string account;
121     mutable messaging::sender atm_queue;
122
123     get_balance(std::string const& account_, messaging::sender atm_queue_):
124         account(account_), atm_queue(atm_queue_)
125     {}
126 };
127
128 struct balance
129 {
130     unsigned amount;
131     explicit balance(unsigned amount_):
132         amount(amount_)
133     {}
134 };
135
```

```
136 struct display_balance
137 {
138     unsigned amount;
139     explicit display_balance(unsigned amount_):
140         amount(amount_)
141     {}
142 };
143
144 struct balance_pressed
145 {};
```

清单C.7 ATM状态机

```
1  class atm
2  {
3      messaging::receiver incoming;
4      messaging::sender bank;
5      messaging::sender interface_hardware;
6
7      void (atm::*state)();
8
9      std::string account;
10     unsigned withdrawal_amount;
11     std::string pin;
12
13     void process_withdrawal()
14     {
15         incoming.wait()
16         .handle<withdraw_ok>(
17             [&](withdraw_ok const& msg)
18             {
19                 interface_hardware.send(
20                     issue_money(withdrawal_amount));
21
22                 bank.send(
23                     withdrawal_processed(account, withdrawal_amount));
24
25                 state=&atm::done_processing;
26             })
27         .handle<withdraw_denied>(
28             [&](withdraw_denied const& msg)
29             {
30                 interface_hardware.send(display_insufficient_funds());
31     }
```

```
32         state=&atm::done_processing;
33     })
34     .handle<cancel_pressed>(
35         [&](cancel_pressed const& msg)
36     {
37         bank.send(
38             cancel_withdrawal(account, withdrawal_amount));
39
40         interface_hardware.send(
41             display_withdrawal_cancelled());
42
43         state=&atm::done_processing;
44     });
45 }
46
47 void process_balance()
48 {
49     incoming.wait()
50     .handle<balance>(
51         [&](balance const& msg)
52     {
53         interface_hardware.send(display_balance(msg.amount));
54
55         state=&atm::wait_for_action;
56     })
57     .handle<cancel_pressed>(
58         [&](cancel_pressed const& msg)
59     {
60         state=&atm::done_processing;
61     });
62 }
63
64 void wait_for_action()
65 {
66     interface_hardware.send(display_withdrawal_options());
67
68     incoming.wait()
69     .handle<withdraw_pressed>(
70         [&](withdraw_pressed const& msg)
71     {
72         withdrawal_amount=msg.amount;
73         bank.send(withdraw(account, msg.amount, incoming));
74         state=&atm::process_withdrawal;
75     })
76     .handle<balance_pressed>(
```

```
77     [&](balance_pressed const& msg)
78     {
79         bank.send(get_balance(account,incoming));
80         state=&atm::process_balance;
81     })
82     .handle<cancel_pressed>(
83     [&](cancel_pressed const& msg)
84     {
85         state=&atm::done_processing;
86     });
87 }
88
89 void verifying_pin()
90 {
91     incoming.wait()
92     .handle<pin_verified>(
93     [&](pin_verified const& msg)
94     {
95         state=&atm::wait_for_action;
96     })
97     .handle<pin_incorrect>(
98     [&](pin_incorrect const& msg)
99     {
100         interface_hardware.send(
101         display_pin_incorrect_message());
102         state=&atm::done_processing;
103     })
104     .handle<cancel_pressed>(
105     [&](cancel_pressed const& msg)
106     {
107         state=&atm::done_processing;
108     });
109 }
110
111 void getting_pin()
112 {
113     incoming.wait()
114     .handle<digit_pressed>(
115     [&](digit_pressed const& msg)
116     {
117         unsigned const pin_length=4;
118         pin+=msg.digit;
119
120         if(pin.length()==pin_length)
121         {
```

```
122         bank.send(verify_pin(account,pin,incoming));
123         state=&atm::verifying_pin;
124     }
125 })
126 .handle<clear_last_pressed>(
127     [&](clear_last_pressed const& msg)
128     {
129         if(!pin.empty())
130         {
131             pin.pop_back();
132         }
133     })
134 .handle<cancel_pressed>(
135     [&](cancel_pressed const& msg)
136     {
137         state=&atm::done_processing;
138     });
139 }
140
141 void waiting_for_card()
142 {
143     interface hardware.send(display_enter_card());
144
145     incoming.wait()
146     .handle<card_inserted>(
147         [&](card_inserted const& msg)
148         {
149             account=msg.account;
150             pin="";
151             interface hardware.send(display_enter_pin());
152             state=&atm::getting_pin;
153         });
154 }
155
156 void done_processing()
157 {
158     interface hardware.send(eject_card());
159     state=&atm::waiting_for_card;
160 }
161
162 atm(atm const&)=delete;
163 atm& operator=(atm const&)=delete;
164 public:
165     atm(messaging::sender bank_,
166         messaging::sender interface_hardware_):
```

```
167     bank(bank_), interface hardware(interface hardware_)
168     {}
169
170     void done()
171     {
172         get_sender().send(messaging::close_queue());
173     }
174
175     void run()
176     {
177         state=&atm::waiting_for_card;
178         try
179         {
180             for(;;)
181             {
182                 (this->*state)();
183             }
184         }
185         catch(messaging::close_queue const&)
186         {
187         }
188     }
189
190     messaging::sender get_sender()
191     {
192         return incoming;
193     }
194 };
```

清单C.8 银行状态机

```
1  class bank_machine
2  {
3      messaging::receiver incoming;
4      unsigned balance;
5  public:
6      bank_machine():
7
8          balance(199)
9      {}
10
11     void done()
12     {
13         get_sender().send(messaging::close_queue());
```

```
14     }
15
16     void run()
17     {
18         try
19         {
20             for(;;)
21             {
22                 incoming.wait()
23                 .handle<verify_pin>(
24                     [&](verify_pin const& msg)
25                     {
26                         if(msg.pin=="1937")
27                         {
28                             msg.atm_queue.send(pin_verified());
29                         }
30                         else
31                         {
32                             msg.atm_queue.send(pin_incorrect());
33                         }
34                     })
35                 .handle<withdraw>(
36                     [&](withdraw const& msg)
37                     {
38                         if(balance>=msg.amount)
39                         {
40                             msg.atm_queue.send(withdraw_ok());
41                             balance-=msg.amount;
42                         }
43                         else
44                         {
45                             msg.atm_queue.send(withdraw_denied());
46                         }
47                     })
48                 .handle<get_balance>(
49                     [&](get_balance const& msg)
50                     {
51                         msg.atm_queue.send(::balance(balance));
52                     })
53                 .handle<withdrawal_processed>(
54                     [&](withdrawal_processed const& msg)
55                     {
56                     })
57                 .handle<cancel_withdrawal>(
58                     [&](cancel_withdrawal const& msg)
```



```
59         {
60         });
61     }
62 }
63 catch(messaging::close_queue const&)
64 {
65 }
66 }
67
68 messaging::sender get_sender()
69 {
70     return incoming;
71 }
72 };
```

清单C.9 用户状态机

```
1  class interface_machine
2  {
3      messaging::receiver incoming;
4  public:
5      void done()
6      {
7          get_sender().send(messaging::close_queue());
8      }
9
10     void run()
11     {
12         try
13         {
14             for(;;)
15             {
16                 incoming.wait()
17                     .handle<issue_money>(
18                     [&](issue_money const& msg)
19                     {
20                         {
21                             std::lock_guard<std::mutex> lk(iom);
22                             std::cout<<"Issuing "
23                                 <<msg.amount<<std::endl;
24                         }
25                     })
26                     .handle<display_insufficient_funds>(
27                     [&](display_insufficient_funds const& msg)
```

```
28         {
29             {
30                 std::lock_guard<std::mutex> lk(iom);
31                 std::cout<<"Insufficient funds"<<std::endl;
32             }
33         })
34     .handle<display_enter_pin>(
35         [&](display_enter_pin const& msg)
36         {
37             {
38                 std::lock_guard<std::mutex> lk(iom);
39                 std::cout<<"Please enter your PIN (0-9)"<<std::endl;
40             }
41         })
42     .handle<display_enter_card>(
43         [&](display_enter_card const& msg)
44         {
45             {
46                 std::lock_guard<std::mutex> lk(iom);
47                 std::cout<<"Please enter your card (I)"
48                     <<std::endl;
49             }
50         })
51     .handle<display_balance>(
52         [&](display_balance const& msg)
53         {
54             {
55                 std::lock_guard<std::mutex> lk(iom);
56                 std::cout
57                     <<"The balance of your account is "
58                     <<msg.amount<<std::endl;
59             }
60         })
61     .handle<display_withdrawal_options>(
62         [&](display_withdrawal_options const& msg)
63         {
64             {
65                 std::lock_guard<std::mutex> lk(iom);
66                 std::cout<<"Withdraw 50? (w)"<<std::endl;
67                 std::cout<<"Display Balance? (b)"
68                     <<std::endl;
69                 std::cout<<"Cancel? (c)"<<std::endl;
70             }
71         })
72     .handle<display_withdrawal_cancelled>(
```

```

73         [&](display_withdrawal_cancelled const& msg)
74     {
75     {
76         std::lock_guard<std::mutex> lk(iom);
77         std::cout<<"Withdrawal cancelled"
78             <<std::endl;
79     }
80     })
81     .handle<display_pin_incorrect_message>(
82     [&](display_pin_incorrect_message const& msg)
83     {
84     {
85         std::lock_guard<std::mutex> lk(iom);
86         std::cout<<"PIN incorrect"<<std::endl;
87     }
88     })
89     .handle<eject_card>(
90     [&](eject_card const& msg)
91     {
92     {
93         std::lock_guard<std::mutex> lk(iom);
94         std::cout<<"Ejecting card"<<std::endl;
95     }
96     });
97     }
98     }
99     catch(messaging::close_queue&)
100     {
101     }
102     }
103
104     messaging::sender get_sender()
105     {
106         return incoming;
107     }
108     };

```

清单C.10 驱动代码

```

1  int main()
2  {
3      bank_machine bank;
4      interface_machine interface_hardware;
5

```

```
6   atm_machine(bank.get_sender(), interface hardware.get_sender());
7
8   std::thread bank_thread(&bank_machine::run, &bank);
9   std::thread if_thread(&interface_machine::run, &interface hardware);
10  std::thread atm_thread(&atm::run, &machine);
11
12  messaging::sender atmqueue(machine.get_sender());
13
14  bool quit_pressed=false;
15
16  while(!quit_pressed)
17  {
18      char c=getchar();
19      switch(c)
20      {
21          case '0':
22          case '1':
23          case '2':
24          case '3':
25          case '4':
26          case '5':
27          case '6':
28          case '7':
29          case '8':
30          case '9':
31          atmqueue.send(digit_pressed(c));
32          break;
33          case 'b':
34          atmqueue.send(balance_pressed());
35          break;
36          case 'w':
37          atmqueue.send(withdraw_pressed(50));
38          break;
39          case 'c':
40          atmqueue.send(cancel_pressed());
41          break;
42          case 'q':
43          quit_pressed=true;
44          break;
45          case 'i':
46          atmqueue.send(card_inserted("acc1234"));
47          break;
48      }
49  }
50
```

```
51     bank.done();
52     machine.done();
53     interface_hardware.done();
54
55     atm_thread.join();
56     bank_thread.join();
57     if_thread.join();
58 }
```