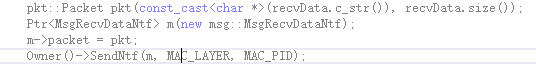
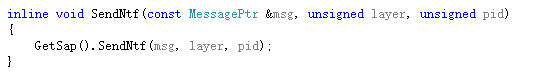
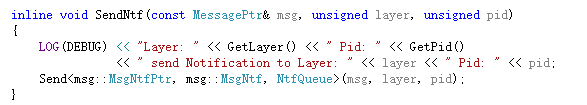
从一个协议层发往另一个协议层的事件，都是通过指针传递的。为了实现多态，HSM通过基类指针传递，这样就带来了如何找到事件原型的问题。例如QPSK中往MAC层发MsgRecvDataNtf事件，生成了指针，传入Module基类中的SendNtf函数。



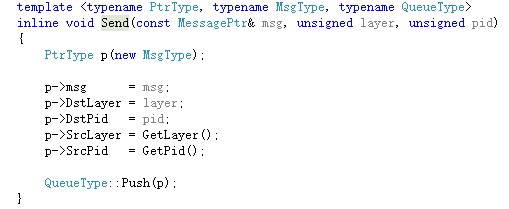
**Module基类中**的函数，需要通过Sap发布事件。



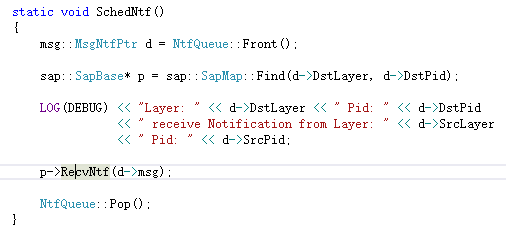
**Sap类中**的函数：



**Sap类中**的Send函数，这里模板参数需要传入：事件本身的指针，层间事件的指针，事件插入的队列。Send函数完成对原始事件的包装（加入原目标层和ID），然后压入层间事件队列的过程。

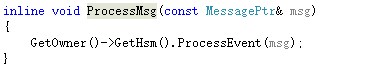


之后就交给调度器处理，**Scheduler类中**的处理函数。这里会找到目标协议层的Sap，然后调用该层的Sap处理函数。（**这里只需要传入原始事件本身即可**）

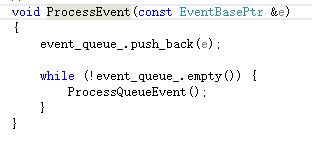


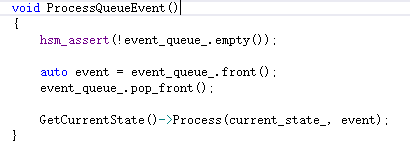
**Sap类中**的处理函数：（事实上，RecvReq和RecvNtf、RecvRsp区别不大，RecvReq会设置协议类中的私有成员：传入事件的层和ID，主要是为了方便协议找到上一层。但**事实上协议直接使用宏来得知上一层**。个人认为这里设置没有必要，Sap中这几个处理函数是一样的，最后协议可以根据原始事件的类型来得知是否需要Rsp。）



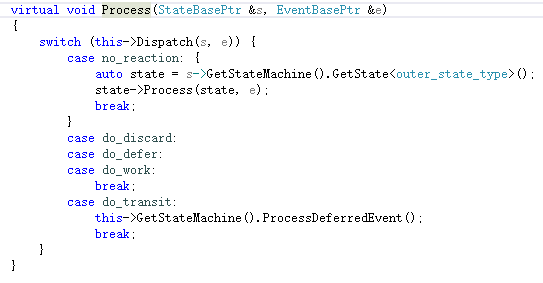


**StateMachine类中**的函数，找到当前状态，然后调用该状态下的处理派遣函数。





**State类中**的函数，需要调用当前的Dispatch函数（StateImpl类中实现）。



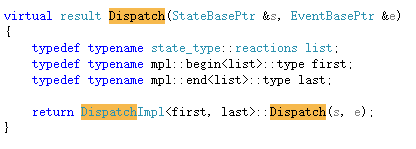
StateImpl类中的函数：

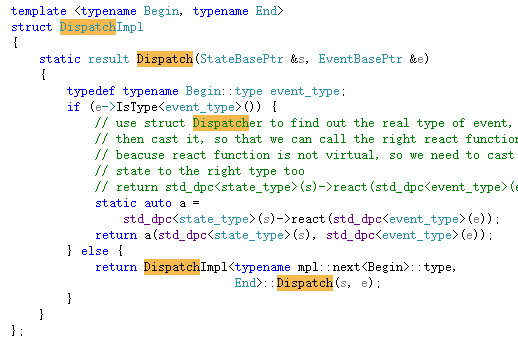
1. 至此，整个过程对原始事件到底是什么仍是未知的，直到在Dispatch函数中调用IsType来查询。

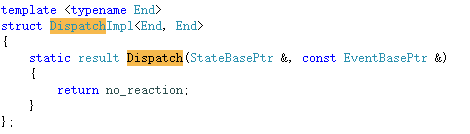
2. 利用DispatchImpl中的Dispatch函数来进行递归查找：先查看第一个事件与传入事件是否匹配，否则把列表中下一个状态作为开始，再生产一个DispatchImpl调用Dispatch函数来查找，如此类推。如果找到了end也没有，就返回no\_reaction。

3. 注意，每个状态都必须先写上一个叫reactions的列表，Dispatch会从这个列表中找是否存在能够处理的事件。

4. 当得知事件类型event\_type之后，就可以将传入的事件强转成该事件原型，然后调用该状态下已经用宏定义好的react函数，然后reactImpl结构体，再调用其中的括号运算符重载，调用需要执行的动作、延时、抛弃、转移等。







每个原始事件都有IsType这个函数。传入模板，然后查看当前事件和传入模板事件是否相同。

