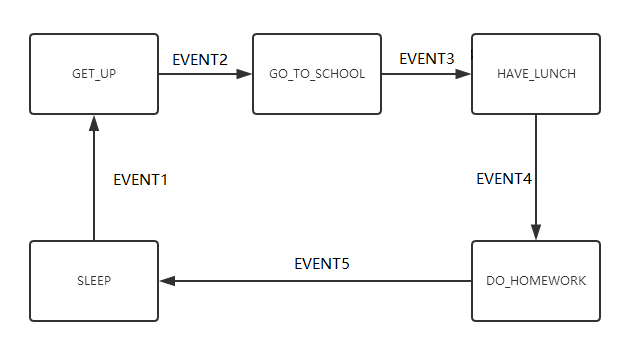
# 1 有限状态机编程

## 单层次有限状态机

### 有限状态机编程思想

使用函数指针实现FSM的思路：建立相应的状态表和动作查询表，根据状态表、事件、动作表定位相应的动作处理函数，执行完成后再进行状态的切换。

### 有限状态机编程例子



FSM的状态转移图

|  |  |
| --- | --- |
| #include <stdio.h>  //#include <windows.h> //windows  #include <unistd.h>  //linux  //比如我们定义了小明一天的状态如下  **enum**  {      SLEEP=0,      GET\_UP,      GO\_TO\_SCHOOL,      HAVE\_LUNCH,      DO\_HOMEWORK,  };  //我们定义的事件有以下几个  **enum**  {      EVENT1 = 1,      EVENT2,      EVENT3,      EVENT4,      EVENT5,  };  **typedef** **struct** FsmTable\_s  {  **int** event;   //事件  **int** CurState;  //当前状态  **void** (\*eventActFun)();  //函数指针  **int** NextState;  //下一个状态  }FsmTable\_t;  **typedef** **struct** FSM\_s  {      FsmTable\_t\* FsmTable;   //指向的状态表  **int** curState;  //FSM当前所处的状态  }FSM\_t;  **int** g\_max\_num;  //状态表里含有的状态个数  **void** GetUp()  {      // do something  **printf**("xiao ming gets up!\n");  }  **void** Go2School()  {      // do something  **printf**("xiao ming goes to school!\n");  }  **void** HaveLunch()  {      // do something  **printf**("xiao ming has lunch!\n");  }  **void** DoHomework()  {      // do something  **printf**("xiao ming does homework!\n");  }  **void** Go2Bed()  {      // do something  **printf**("xiao ming goes to bed!\n");  }  /\*状态机注册\*/  **void** FSM\_Regist(FSM\_t\* pFsm, FsmTable\_t\* pTable)  {      pFsm->FsmTable = pTable;  }    /\*状态迁移\*/  **void** FSM\_StateTransfer(FSM\_t\* pFsm, **int** state)  {      pFsm->curState = state;  }   /\*事件处理\*/  **void** FSM\_EventHandle(FSM\_t\* pFsm, **int** event)  {  **int** i = 0;      FsmTable\_t\* pActTable = pFsm->FsmTable;  **void** (\*eventActFun)() = NULL;  //函数指针初始化为空  **int** NextState;  **int** CurState = pFsm->curState;  **int** flag = 0; //标识是否满足条件      /\*获取当前动作函数\*/  **for** (; i<g\_max\_num; i++)      {          //当且仅当当前状态下来个指定的事件，我才执行它  **if** (event == pActTable[i].event && CurState == pActTable[i].CurState)          {              flag = 1;              eventActFun = pActTable[i].eventActFun;              NextState = pActTable[i].NextState;  **break**;          }      }  **if** (flag) //如果满足条件了      {          /\*动作执行\*/  **if** (eventActFun)          {              eventActFun();          }          //跳转到下一个状态          FSM\_StateTransfer(pFsm, NextState);      }  **else**      {          // do nothing      }  }  FsmTable\_t XiaoMingTable[] =  {      //{到来的事件，当前的状态，将要要执行的函数，下一个状态}      { EVENT1,  SLEEP,           GetUp,        GET\_UP },      { EVENT2,  GET\_UP,          Go2School,    GO\_TO\_SCHOOL },      { EVENT3,  GO\_TO\_SCHOOL,    HaveLunch,    HAVE\_LUNCH },      { EVENT4,  HAVE\_LUNCH,      DoHomework,   DO\_HOMEWORK },      { EVENT5,  DO\_HOMEWORK,     Go2Bed,       SLEEP },        //add your codes here  };  //初始化FSM  **void** InitFsm(FSM\_t\* pFsm)  {      g\_max\_num = **sizeof**(XiaoMingTable) / **sizeof**(FsmTable\_t);      pFsm->curState = SLEEP;      FSM\_Regist(pFsm, XiaoMingTable);  }  //测试用的  **void** test(**int** \*event)  {  **if** (\*event == EVENT5)      {          \*event = EVENT1;      }  **else**      {          (\*event)++;      }  }  **int** main()  {      FSM\_t fsm;      InitFsm(&fsm);  **int** event = EVENT1;      //小明的一天,周而复始的一天又一天，进行着相同的活动  **while** (1)      {  **printf**("fsm current state %d\n", fsm.curState);  **printf**("event %d is coming...\n", event);          FSM\_EventHandle(&fsm, event);          test(&event);          Sleep(1000);  //休眠1秒，方便观察      }  **return** 0;  } |  |