## FSM finte-state machine 有限状态机

表示在有限个状态，以及在这些状态之间的转移的动作等行为的数学模型。

FSM通常包含几个要素，状态的管理，状态的监控，状态的触发，状态触发的引起的动作。

State(S) Event(E)->Actions(A) State(S+)

当前状态是S，发生了事件E，执行了操作A，然后状态转换为S+。

状态机：理解就是输出取决于输入与当前状态。

有限状态机可以确定状态的个数，比如人物的状态。当前状态是无限的，不能确定其个数，比如人物的位置。

## 状态机案例

这个状态机很垃圾，为啥，因为他是根据event和state这两个值去遍历这个数组才确定我执行哪一个函数的，垃圾。

就该用朋哥说的那个，二维数组，一维数组表示状态，二维数组里面是事件，数组元素是函数指针即可。

小明的一天活动状态例子

enum { GET\_UP,

GO\_TO\_SCHOOL,

HAVE\_LUNCH,

DO\_HOMEWORK,

SLEEP, };

定义小明的一天状态的枚举。

enum {

EVENT1 = 1,

EVENT2,

EVENT3, };

定义小明的一个的事件。

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;

定义状态表的数据结构

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}

{ EVENT2, SLEEP, GetUp, GET\_UP },

{ EVENT1, GET\_UP, Go2School, GO\_TO\_SCHOOL },

{ EVENT2, GO\_TO\_SCHOOL, HaveLunch, HAVE\_LUNCH },

{ EVENT3, HAVE\_LUNCH, DoHomework, DO\_HOMEWORK }

{ EVENT1, DO\_HOMEWORK, Go2Bed, SLEEP }, //add your codes here };

可以看到这是上面的那个结构体的数组，我们在mme的S1AP代码李貌似看到过类似的，也是一个结构体数组，这个结构体里面保存了状态，事件和函数指针之间的对应关系。

下面我们来看具体的实现。

状态机注册，咋说呢，我们也知道状态机其实是用来定义规则逻辑的，还有一个地方用来保存数据，比如mme中的usercontext，用户上下文，用户上下文中肯定就会有一个数据来保存当前状态。

利用当前的保存的用户状态数据，和收到的事件，和我们定义的状态机数据，那么就可以进行状态机的操作。

这里的意思是呢，把保存的用户数据和这个逻辑数组放在一起，弄成一个结构体，叫做FSM\_t

这个结构体有啥好处呢，这个结构体主体是用户数据，这个结构体李还有有一个指针，指向对应的逻辑数组，这样的话如果你想把整个状态机都换了也是很方便的。

/\*状态机注册\*/

所谓的状态机注册呢，就是给这个用户上下文指定状态机逻辑数组。

void FSM\_Regist(FSM\_t\* pFsm, FsmTable\_t\* pTable)

{

    pFsm->FsmTable = pTable;

}

/\*状态迁移\*/

状态迁移函数，就是修改下状态呗，可能会需要加锁啥的？？？？？？

void FSM\_StateTransfer(FSM\_t\* pFsm, int state)

{

    pFsm->curState = state;

}

/\*事件处理\*/

void FSM\_EventHandle(FSM\_t\* pFsm, int event)

{

FsmTable\_t\* pActTable = pFsm->FsmTable;

    这里是先弄出一个函数指针，看到了么，函数指针的定义方式

返回值（\*函数名）（参数列表）

是在函数名前面顶了一个\*，这个就是函数指针。

但是这里要注意，这里的\*是声明用的，真正用函数指针执行的时候就用这个名字就可以了。

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

    int i;

    /\*获取当前动作函数\*/

    for (i = 0; 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\*/

看到了么，函数指针的执行。

        if (eventActFun)

        {

            eventActFun();

        }

        //跳转到下一个状态

        FSM\_StateTransfer(pFsm, NextState);

    }

    else

    {

        // do nothing

    }

}

int main()

{

FSM\_t fsm;

定义上面我们说的那个结构体

InitFsm(&fsm);

初始化，可能就是设置FSM\_t结构体的值和注册状态机

    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("event %d is coming...\n", event);

        FSM\_EventHandle(&fsm, event);

        printf("fsm current state %d\n", fsm.curState);

        test(&event);

        sleep(1);  //休眠1秒，方便观察

    }

    return 0;

}

## 状态保护定时器

就是一个用户上下文，不能长时间处于某一个状态，需要开一个定时器来保护。

## 一段式，二段式，三段式