Aquarius (其中还有英文网站)
[编辑]高手&大师
ammana_babi
Richard Stallman
Roland McGrath
steedhorse
taodm
侯捷
周立功
徐艺波个人网站
艺术编程
陈莉君
[编辑]好书推荐
C++学习推荐书目
c 语言的提高
[编辑]好友
袁东
存档
2010 年 05 月(8)
2010 年 04 月(2)
2010 年 01 月(5)
2009 年 08 月(1)
2009 年 06 月(1)
2009 年 05 月(4)
2009 年 04 月(5)
2009 年 03 月(8)
2009 年 02 月(1)
2009 年 01 月(1)
2008 年 05 月(21)
2008 年 04 月(3)
公告：
CSDN 产品事业部开设官方博客了！来关注我们的一举一动吧！[意见反馈][官方博客] C 语言实现有限状态机 收藏
以下是转载内容：

☆————————————————————————————传说中的分隔符————————————————————————————————————☆

来源 1：http://www.cnblogs.com/swingboat/archive/2005/07/27/201488.html

【转载 1】有限状态机的实现 < type="text/javascript">

有限状态机（Finite State Machine 或者 Finite State Automata)是软件领域中一种重要的工具，很多东西的模型实际上就是有限状态机。

最近看了一些游戏编程 AI 的材料，感觉游戏中的 AI，第一要说的就是有限状态机来实现精灵的 AI，然后才是 A*寻路，其他学术界讨论比较多的神经网络、模糊控制等问题还不是很热。

FSM 的实现方式：

1） switch/case 或者 if/else

这无意是最直观的方式，使用一堆条件判断，会编程的人都可以做到，对简单小巧的状态机来说最合适，但是毫无疑问，这样的方式比较原始，对庞大的状态机难以维护。

2） 状态表

维护一个二维状态表，横坐标表示当前状态，纵坐标表示输入，表中一个元素存储下一个状态和对应的操作。这一招易于维护，但是运行时间和存储空间的代价较大。

3） 使用 State Pattern

使用 State Pattern 使得代码的维护比 switch/case 方式稍好，性能上也不会有很多的影响，但是也不是 100％完美。不过 Robert C. Martin 做了两个自动产生 FSM 代码的工具，for java 和 for C++各一个，在 http://www.objectmentor.com/resources/index 上有免费下载，这个工具的输入是纯文本的状态机描述，自动产生符合 State Pattern 的代码，这样 developer 的工作只需要维护状态机的文本描述，每必要冒引入 bug 的风险去维护 code。

4） 使用宏定义描述状态机

一般来说，C++编程中应该避免使用#define，但是这主要是因为如果用宏来定义函数的话，很容易产生这样那样的问题，但是巧妙的使用,还是能够产生奇妙的效果。MFC 就是使用宏定义来实现大的架构的。

在实现 FSM 的时候，可以把一些繁琐无比的 if/else 还有花括号的组合放在宏中，这样，在代码中可以 3）中状态机描述文本一样写，通过编译器的预编译处理产生 1）一样的效果，我见过产生 C 代码的宏，如果要产生 C++代码，己软 MFC 可以，那么理论上也是可行的。

【评】：状态表的实现方法，《C 专家编程》第 8 章有具体说明，转载【6】

☆————————————————————————传说中的分隔符————————————————————————☆

来源 2：http://hi.baidu.com/juneshine/blog/item/6bff718bd5902f13c9fc7a14.html

【转载 2】有限状态机的 c 实现

2007-05-11 15:12

網絡上可以搜索到很多有限狀態機的代碼和理論分析，這兒僅僅是做一個簡單的例子，僅供入門參考。

这儿以四位密码校验作为状态机的例子，连续输入 2479 就可以通过密码测试。一个非常简单的例子，在实际的状态机实例中，状态转移表要更復雜一些，不過方式非常類似。在狀態查詢的地方可以做優化，同時對于輸入量也可以做有效性優化。具體代碼如下：

view plaincopy to clipboardprint?

c.h

```
typedef enum{
STATE1 = 1,
STATE2,
STATE3,
STATE4,
STATE5,//password pass
//...ADD here
}STATE;

typedef enum{
INPUT1 = '2',
INPUT2 = '4',
INPUT3 = '7',
INPUT4 = '9',
}INPUT;

typedef struct
{
STATE cur_state;
INPUT input;
STATE next_state;
}STATE_TRANS;
```

c.h

```
typedef enum{
STATE1 = 1,
STATE2,
STATE3,
STATE4,
STATE5,//password pass
```

```
//...ADD here
}STATE;

typedef enum{
INPUT1 = '2',
INPUT2 = '4',
INPUT3 = '7',
INPUT4 = '9',
}INPUT;

typedef struct
{
STATE cur_state;
INPUT input;
STATE next_state;
}STATE_TRANS;
```

c.c

```
#include     <stdio.h>
#include "c.h"

STATE_TRANS state_trans_arry[] =
{
{STATE1,INPUT1,STATE2},
{STATE2,INPUT2,STATE3},
{STATE3,INPUT3,STATE4},
{STATE4,INPUT4,STATE5},
};
#define STATE_TRANS_CNT (sizeof(state_trans_arry)/sizeof(state_trans_arry[0]))

int main()
{
int i;
char ch;
STATE state_machine = STATE1;

while(ch != 'e')
{
    ch = getchar();
    if((ch >= '0') && (ch <= '9'))//for digit password input only
    {
```

```
        for(i = 0;i < STATE_TRANS_CNT;i++)
        {
         if((ch == state_trans_arry[i].input) && (state_machine == state_trans_arry[i].cur_state))
         {
          state_machine = state_trans_arry[i].next_state;
          continue;
         }
         else if(i == (STATE_TRANS_CNT - 1))//no transfer match,reset state
         {
          state_machine = STATE1;
         }
        }
        if(state_machine == STATE5)
          printf("Password correct,state transfer machine pass!\n");
      }
}
return 0;
}
```

c.c

```
#include     <stdio.h>
#include "c.h"

STATE_TRANS state_trans_arry[] =
{
{STATE1,INPUT1,STATE2},
{STATE2,INPUT2,STATE3},
{STATE3,INPUT3,STATE4},
{STATE4,INPUT4,STATE5},
};
#define STATE_TRANS_CNT (sizeof(state_trans_arry)/sizeof(state_trans_arry[0]))

int main()
{
int i;
char ch;
STATE state_machine = STATE1;

while(ch != 'e')
{
     ch = getchar();
     if((ch >= '0') && (ch <= '9'))//for digit password input only
     {
       for(i = 0;i < STATE_TRANS_CNT;i++)
```

```
        {
         if((ch == state_trans_arry[i].input) && (state_machine == state_trans_arry[i].cur_state))
         {
          state_machine = state_trans_arry[i].next_state;
          continue;
         }
         else if(i == (STATE_TRANS_CNT - 1))//no transfer match,reset state
         {
          state_machine = STATE1;
         }
        }
        if(state_machine == STATE5)
         printf("Password correct,state transfer machine pass!\n");
      }
}
return 0;
}
```

【评】: 在 VC6 下运行该程序并没有达到目的，即连续输入字符 2479 也没有任何输出信息，个人根据转载第一遍文章的 FSM 的实现的第一种方法，见【原创之源程序】

☆————————————————————————传说中的分隔符————————————————————————☆

来源 3：http://lionwq.spaces.eepw.com.cn/articles/article/item/16363

【转载 3】有限状态机自动机

状态图－－一个图的数据结构！
1.while + switch;
2.状态机：就是指定系统的所有可能的状态及状态间跳转的条件，然后设一个初始状态输入给这台机器，机器就会自动运转，或最后处于终止状态，或在某一个状态不断循环。
游戏中状态切换是很频繁的。 可能以前要切换状态就是 if~else，或者设标志，但这些都不太结构化， 如果把它严格的设为一种标准的状态机，会清楚的多。

比如控制一扇门的运动， 初始时门是关的， 当有力作用在门上时， 门开始慢慢打开，力的作用完后，门渐渐停止不动， 当有反向的力时，门又渐渐关上， 知道回到初始关的状态。这个你会怎么来编程实现呢。 似乎很麻烦， 的确，没有状态机的思想时会很烦，设很多标志，一堆 if 条件。
用状态机的话，不只是代码更清晰， 关键是更符合逻辑和自然规律， 不同状态不同处理，

满足条件则跳转到相关状态。

伪码如下：

```
enum
{
 CLOSED, //  关上状态
 CLOSING, //  正在关状态
 OPENED, //  打开状态
 OPENING, //  正在开的状态
}doorState = CLOSED; //  初始为关

Update()
{
       switch(doorState)
       case CLOSED:
              if (有人推门)
                     doorState = OPENING; //  跳转到正在开状态
       break;
       case OPENING:
              door.Rotation += DeltaAngle; //  门的旋转量递增
              if (门的速度为零) / /  力的作用已去
                     doorState = OPENED; //  跳转到开状态
     break;
     case OPENED:
            if (有人关门)
                   doorState = CLOSING;
   break;
   case CLOSING:
            door.Rotation -= DeltaAngle; //  门的旋转量递减
            if (门的旋转角度减为零)
                   doorState = CLOSED; //  门已关上
     break;
}

//  而绘制代码几乎不用怎么变, 门就是会严格按照状态机的指示去运动，  该停就会停
Render()
{
       RotateView(door.Rotation);
       DrawDoor(door.Position);
}
enum
{
```

```
 CLOSED, //  关上状态
 CLOSING, //  正在关状态
 OPENED, //  打开状态
 OPENING, //  正在开的状态
}doorState = CLOSED; //  初始为关

Update()
{
      switch(doorState)
      case CLOSED:
            if (有人推门)
                  doorState = OPENING; //  跳转到正在开状态
      break;
      case OPENING:
            door.Rotation += DeltaAngle; //  门的旋转量递增
            if (门的速度为零) / /  力的作用已去
                  doorState = OPENED; //  跳转到开状态
     break;
     case OPENED:
           if (有人关门)
                 doorState = CLOSING;
    break;
    case CLOSING:
           door.Rotation -= DeltaAngle; //  门的旋转量递减
           if (门的旋转角度减为零)
                 doorState = CLOSED; //  门已关上
     break;
}

//  而绘制代码几乎不用怎么变, 门就是会严格按照状态机的指示去运动，  该停就会停
Render()
{
      RotateView(door.Rotation);
      DrawDoor(door.Position);
}
```

这是一个简单但很典型的例子， 状态机的应用太多了。
就说一个基本游戏的运转: （用到了一个状态然后还有子状态）

```
UpdateGame()
BEGIN;
    switch(gameState)
    case  等待选择菜单: //它有三个子状态。
```

```
    if (选择菜单项 == 开始)
    {
          游戏初始；
        gameState = 开始游戏
    }
    if (选择菜单项 == 选项)
        gameState = 设置
    if (选择菜单项 == 退出)
          gameState = 退出

  case 开始：

  游戏运行；
  if (用户按退出键)
  gameState = 等待选择菜单 ;
  ...其他的状态跳转处理;
  case 退出:
  释放资源；
  退出；
  case 设置:
  分别处理不同的选项， 跳转不同的子状态；
  case .... // 其他状态的处理

END;
UpdateGame()
BEGIN;
    switch(gameState)
    case 等待选择菜单: //它有三个子状态。
    if (选择菜单项 == 开始)
    {
          游戏初始；
        gameState = 开始游戏
    }
    if (选择菜单项 == 选项)
        gameState = 设置
    if (选择菜单项 == 退出)
          gameState = 退出

  case 开始：

  游戏运行；
  if (用户按退出键)
  gameState = 等待选择菜单 ;
  ...其他的状态跳转处理;
```

case 退出:
释放资源；
退出；
case 设置:
分别处理不同的选项， 跳转不同的子状态；
case .... // 其他状态的处理

END;

某一个状态可以包含更多的子状态， 这样最好是同一层次的状态设为一个枚举, 并分到另一个 switch 处理
如 enum STATES{state1, state2, state3}; state2 又包含若干状态
则再定义 enum SUB_STATE2{sub_state2_1, sub_state2_2, sub_state2_3,};

想很多基本的渲染效果， 如淡入淡出， 闪烁等等， 用状态的思想会事半功倍， 思路也更清晰。

其实像 Opengl, Direct3D 这样的渲染引擎本身就是状态机， 当你设置渲染的状态， 这台机器就保持这个状态进行渲染工作，如保持光照位置，保持片元颜色， 直到你再次改变它的状态。

状态机的应用实在太广， 相关理论也很多， 最近上课学的随机过程里也讲到一些， 数字电路里的时序逻辑器件也是用状态机来描述。 这些不必多说了。

总之， 用状态机的角度去看待问题， 往往能把比较复杂的系统分解为能单独处理的众多子状态， 处理也会得心应手。希望大家多用它， 很好的东西。

二、
推荐这个：[程序员杂志 2004.8 月刊_state 模式和 composite 模式实现的状态机引擎]
http://www.contextfree.net/wangyw/source/oofsm.html

个人感觉状态机的几个不同实现阶段：
1、switch/case 最原始的实现方式,是很多的 c 程序员习惯采用的方式。

2、查找表[状态、事件、动作],稍微做了一点改进。有点类似 MFC 的雏形。

3、在以上基础上做的一些改进或者变体。
[比如用一个栈结构，激活的状态位于栈顶，自动的映射事件和动作的对应，再或者通过一些巧妙的宏等手段进行包装。但是线性结构在实际中使用比较受限、过于技巧性的宏比较难于理解...]

4、面向对象的设计下、灵活运用模式。如上面给出的链接。重用性和灵活性方面都有不错的表现。沿袭类似的设计思路、根据实际开发内容进行改造后利用。

【评】：伪代码部分，可以帮助很好的理解【转载 1】文章中叙述的 FSM 的实现方法 1；查找表[状态、事件、动作],稍微做了一点改进。有点类似 MFC 的雏形类似于【转载 1】文章中叙述的 FSM 的实现方法 2（状态表）

☆————————————————————————传说中的分隔符————————————————————————☆

来源 4：http://hi.baidu.com/yorkbluedream/blog/item/620075faa630db1ba8d31192.html

【转载 4】fsm implemented in C code(FSM 状态机用 C 实现)

用 C 语言实现一个状态机，很简单，和大家分享
这是我做毕业设计时，用 nRF24L01 组建了一个简单的网络，做的一个小的状态机，网络中三个节点，开始拓扑为网状，后来为星型。

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>


//Finite state machine declaration
//state declaration
#define IDLE 0 //idle state in rx mode
#define M_BROADCAST 1 //broadcast state in tx mode,broadcast to be a master point
#define M_WAIT_BROADCAST_ACK 2 //wait for broadcast ack state in rx mode,wait for the point ack in a specific time window
#define M_WAIT_COMMAND 3 //wait for command state,wait for PC command via UART
#define M_BROADCAST_CANCEL 4 //broadcast cancel state,broadcast to cancel master point

#define S_BROADCAST_ACK 5 //slave mode,send back self physical address
#define S_WAIT_COMMAND 6 //slave mode, wait for command from the master point

//state transition trig
//used in master mode
int isReqBeMaster = 0;//Is PC request the point to be master?
int isTimeout = 0;//Is time out?
int isReqCancelMaster = 0;//Is request to cancel master?
```

//used in slave mode

```
int isRxBroadcast = 0;//Is there a point broadcast to be master?
int isRxBroadcastCancel = 0;//Is receive broadcast cancel master?

typedef struct fsmtag
{
	int state; //state
	int timeouttime; //time out time in milliseconds
}fsm;

//function prototype

int main()
{
	fsm f;

	f.state = IDLE;
	f.timeouttime = 0;

	while(1)
	{
		switch(f.state)
		{
		case IDLE:
			puts("IDLE\nWait for isReqBeMaster(1/0) isRxBroadcast(1/0):");
			scanf("%d %d",&isReqBeMaster,&isRxBroadcast);
			if(isReqBeMaster)
			{
				f.state = M_BROADCAST;
				break;
			}
			else if(isRxBroadcast)
			{
				f.state = S_BROADCAST_ACK;
				break;
			}
			else
				break;
		case M_BROADCAST:
			puts("M_BROADCAST\nBroadcasting...\n");
			f.state = M_WAIT_BROADCAST_ACK;
		case M_WAIT_BROADCAST_ACK:
			puts("M_WAIT_BROADCAST_ACK\nWaiting for isTimeout(1/0):");
			scanf("%d",&isTimeout);
			if(isTimeout)
```