C語言實現簡單有限狀態機(FSM)

原創 stpeace 2019-04-13 18:38

轉載地址：https://www.jianshu.com/p/917c0fb8778b?utm_campaign=maleskine&utm_content=note&utm_medium=seo_notes&utm_source=recommendation

有限狀態機（finite state machine）簡稱FSM，表示有限個狀態及在這些狀態之間的轉移和動作等行爲的數學模型，在計算機領域有着廣泛的應用。FSM是一種邏輯單元內部的一種高效編程方法，在服務器編程中，服務器可以根據不同狀態或者消息類型進行相應的處理邏輯，使得程序邏輯清晰易懂。

狀態機實現的方式有多種，下面講述三種.

1.使用if/else if語句實現的FSM

使用if/else if語句是實現的FSM最簡單最易懂的方法，我們只需要通過大量的if /else if語句來判斷狀態值來執行相應的邏輯處理。

看看下面的例子:

S stpeace

```c
#include <stdio.h>
enum year_state
{
    SPRING,
    SUMMER,
    AUTUMN,
    WINTER
};

void spring_thing()
{
    printf("hello spring\n");
}
void summer_thing()
{
    printf("hello summer\n");
}
void autumn_thing()
{
    printf("hello autumn\n");
}
void winter_thing()
{
    printf("hello winter\n");
}

int main()
{
    int state = SPRING;
    while (1)
    {
        if (state == SPRING)
        {
            spring_thing();//相應的處理
            state = SUMMER;//狀態改變
        }
        else if (state == SUMMER)
        {
            summer_thing();
            state = AUTUMN;
        }
        else if (state == AUTUMN)
        {
            autumn_thing();
            state = WINTER;
        }
        else if (state == WINTER)
        {
            winter_thing();
            state = SPRING;
        }
        sleep(1);
    }

    return 0;
}
```

簡單易懂,這裏實現了四季的更替,因爲只有四種狀態,所以邏輯清楚,試想如果有個幾十種狀態,我們的if else將會變得十分之長,維護起來很麻煩,刪減和添加狀態變得不方便.但是通過這個例子我們認識到狀態機的內涵.

如下圖:

最新文章

VOA慢速英語精聽極致練習
濤哥讀情詩：What it is like to Love You
數據庫連接池的原理及驗證
暫停筆試面試相關的內容
從最近三個實際問題來看tcpdump抓包

最新評論文章

[2022] Top Rated CheckPoint 156-315.80 Exam Questions
Qualified Writing Service in Australia can allow students to achieve better grades
Take assignment helper to resolve the paper queries easily
Updated CompTIA DA0-001 Exam Questions (2022)
美國黑金效果和其它速效藥的不同之處
Microsoft DP-500 PDF Question [2022]-Secret To Pass Exam In First Attempt-[PremiumDumps]

在狀態1時,遇到一個事件,此刻發生狀態轉換,一般在狀態轉換前,先要進行事件的處理,然後改變狀態位.然後進入狀態2,以此類推.

2.使用switch case

這種做法和if else類似,結構上更清楚一些,代碼如下:

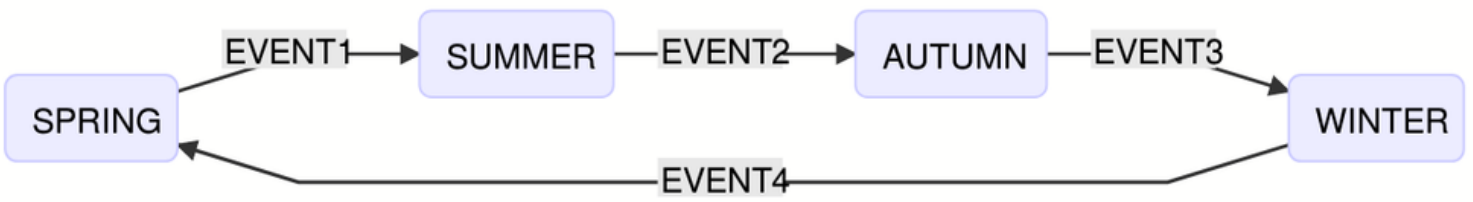
```
int main()
{
    int state = SPRING;
    while (1)
    {
        switch(state){
        case SPRING:
            spring_thing();
            state = SUMMER;
            break;
        case SUMMER:
            summer_thing();
            state = AUTUMN;
            break;
        case AUTUMN:
            autumn_thing();
            state = WINTER;
            break;
        case WINTER:
            winter_thing();
            state = SPRING;
            break;
        default:
            break;

        }
        sleep(1);
    }

    return 0;
}
```

3.函數指針實現FSM

使用函數指針實現FSM的思路：建立相應的狀態表和動作查詢表，根據狀態表、事件、動作表定位相應的動作處理函數，執行完成後再進行狀態的切換。

當然使用函數指針實現的FSM的過程還是比較費時費力，但是這一切都是值得的，因爲當你的程序規模大時候，基於這種表結構的狀態機，維護程序起來也是得心應手。

首先我們畫出這個表



代碼關鍵部分:

1.定義狀態數據的枚舉類型

```
enum year_state
```

2.定義事件的枚舉類型

```
enum year_event
{
    EVENT1 = 1,
    EVENT2,
    EVENT3,
    EVENT4,
};
```

3.定義狀態表的數據類型

```
typedef struct FsmTable_s
{
    int event;   //事件
    int CurState;  //當前狀態
    void (*eventActFun)();  //函數指針
    int NextState;  //下一個狀態
}FsmTable_t;
```

4.定義處理函數及建立狀態表

```
void spring_thing()
{
    printf("this is spring\n");
}
void summer_thing()
{
    printf("this is summer\n");
}
void autumn_thing()
{
    printf("this is autumn\n");
}
void winter_thing()
{
    printf("this is winter\n");
}

FsmTable_t year_table[] =
{
    //{到來的事件，當前的狀態，將要要執行的函數，下一個狀態}
    { EVENT1,  SPRING,    summer_thing,  SUMMER },
    { EVENT2,  SUMMER,    autumn_thing,  AUTUMN },
    { EVENT3,  AUTUMN,    winter_thing,  WINTER },
    { EVENT4,  WINTER,    spring_thing,  SPRING },
    //add your codes here
};
```

5.狀態機類型,及狀態機接口函數

```
/*狀態機類型*/
typedef struct FSM_s{
    int curState;//當前狀態
    FsmTable_t * pFsmTable;//狀態表
    int size;//表的項數
}FSM_t;

/*狀態機註冊,給它一個狀態表*/
void FSM_Regist(FSM_t* pFsm, FsmTable_t* pTable)
{
    pFsm->pFsmTable = pTable;
}

/*狀態遷移*/
void FSM_StateTransfer(FSM_t* pFsm, int state)
{
    pFsm->curState = state;
}

/*事件處理*/
void FSM_EventHandle(FSM_t* pFsm, int event)
{
    FsmTable_t* pActTable = pFsm->pFsmTable;
    void (*eventActFun)() = NULL;  //函數指針初始化爲空
    int NextState;
    int CurState = pFsm->curState;
    int g_max_num = pFsm->size;
    int flag = 0; //標識是否滿足條件
    int i;

    /*獲取當前動作函數*/
    for (i = 0; i<g_max_num; i++)
    {
        //當且僅當當前狀態下來個指定的事件，我才執行它
        if (event == pActTable[i].event && CurState == pActTable[i].CurState)
        {
            flag = 1;
            eventActFun = pActTable[i].eventActFun;
            NextState = pActTable[i].NextState;
            break;
        }
    }


    if (flag) //如果滿足條件了
    {
        /*動作執行*/
        if (eventActFun)
        {
            eventActFun();
        }

        //跳轉到下一個狀態
        FSM_StateTransfer(pFsm, NextState);
    }
    else
    {
        printf("there is no match\n");
    }
}
```

測試程序代碼爲:

```c
/*state.c*/
#include <stdio.h>


enum year_state{
    SPRING = 1,
    SUMMER,
    AUTUMN,
    WINTER
};



enum year_event{
    EVENT1 = 1,
    EVENT2,
    EVENT3,
    EVENT4,
};


typedef struct FsmTable_s{
    int event;   //事件
    int CurState;  //當前狀態
    void (*eventActFun)();  //函數指針
    int NextState;  //下一個狀態
}FsmTable_t;




void spring_thing()
{
    printf("this is spring\n");
}
void summer_thing()
{
    printf("this is summer\n");
}
void autumn_thing()
{
    printf("this is autumn\n");
}
void winter_thing()
{
    printf("this is winter\n");
}


FsmTable_t year_table[] =
{
    //{到來的事件，當前的狀態，將要要執行的函數，下一個狀態}
    { EVENT1,  SPRING,    summer_thing,  SUMMER },
    { EVENT2,  SUMMER,    autumn_thing,  AUTUMN },
    { EVENT3,  AUTUMN,    winter_thing,  WINTER },
    { EVENT4,  WINTER,    spring_thing,  SPRING },
    //add your codes here
};




typedef struct FSM_s{
    int curState;//當前狀態
    FsmTable_t * pFsmTable;//狀態表
    int size://表的項數
```

```
}

/*狀態遷移*/
void FSM_StateTransfer(FSM_t* pFsm, int state)
{
    pFsm->curState = state;
}

/*事件處理*/
void FSM_EventHandle(FSM_t* pFsm, int event)
{
    FsmTable_t* pActTable = pFsm->pFsmTable;
    void (*eventActFun)() = NULL;  //函數指針初始化爲空
    int NextState;
    int CurState = pFsm->curState;
    int g_max_num = pFsm->size;
    int flag = 0; //標識是否滿足條件
    int i;

    /*獲取當前動作函數*/
    for (i = 0; i<g_max_num; i++)
    {
        //當且僅當當前狀態下來個指定的事件，我才執行它
        if (event == pActTable[i].event && CurState == pActTable[i].CurState)
        {
            flag = 1;
            eventActFun = pActTable[i].eventActFun;
            NextState = pActTable[i].NextState;
            break;
        }
    }


    if (flag) //如果滿足條件了
    {
        /*動作執行*/
        if (eventActFun)
        {
            eventActFun();
        }

        //跳轉到下一個狀態
        FSM_StateTransfer(pFsm, NextState);
    }
    else
    {
        printf("there is no match\n");
    }
}


int main()
{
    FSM_t year_fsm;
    FSM_Regist(&year_fsm,year_table);
    year_fsm.curState = SPRING;
    year_fsm.size = sizeof(year_table)/sizeof(FsmTable_t);


    printf("\n-------1--init spring------\n");
    printf("state:%d\n",year_fsm.curState);

    printf("\n-------2--spring->summer------\n");
    FSM_EventHandle(&year_fsm,EVENT1);
    printf("state:%d\n",year_fsm.curState);

    printf("\n-------3--summer->autumn------\n");
    FSM_EventHandle(&year_fsm,EVENT2);
    printf("state:%d\n",year_fsm.curState);
```

```
    FSM_EventHandle(&year_fsm,EVENT4);
    printf("state:%d\n",year_fsm.curState);

    printf("\n-------6--receive EVENT2 not EVENT1------\n");
    FSM_EventHandle(&year_fsm,EVENT2);
    printf("state:%d\n",year_fsm.curState);

    return 0;
}
```

結果爲:

```
-------1--init spring------
state:1

-------2--spring->summer------
this is summer
state:2

-------3--summer->autumn------
this is autumn
state:3

-------4--autumn->winter------
this is winter
state:4

-------5--winter->spring------
this is spring
state:1

-------6--receive EVENT2 not EVENT1------
there is no match
state:1
```

【海濤客】海苔啾咪捲

登錄以後才評論...

登 录

所有評論

還沒有人評論，想成為第一個評論的人麼? 請在上方評論欄輸入並且點擊發布.

相關文章

ios使用word-break:break-all無效

記錄一下 移動端 IOS 使用word-break:break-all無效(安卓，模擬器正常)，改爲word-break:break-word

指尖流年1218 2022-10-18 14:33:52

包裝類與數組

包裝類（Wrapper Class）。 針對原生數據類型的包裝。 包裝類（8 個）都位於 java.lang 包下。 java 中的 8 個包裝類分別是：Byte，Short，Integer，Long，Float，Double，Cha

燈塔下的守望者 2022-10-18 14:32:12

Oracle查看數據庫版本等信息

查看數據庫版本 -- 查看oracle版本 select * from product_component_version; 查看數據庫列表 -- 查看數據庫列表 select username as schema_name from s

燈塔下的守望者 2022-10-18 14:32:12

Java開發學習(三十八)----SpringBoot整合junit Java開發學習(三十五)----SpringBoot快速入門及起步依賴解析

先來回顧下 Spring 整合 junit @RunWith(SpringJUnit4ClassRunner.class) @ContextConfiguration(classes = SpringConfig.class) public

|舊市拾荒| 2022-10-18 14:30:42

開源WindivertDotnet

0 前言 Hi，好久沒有寫博客，因爲近段時間沒有新的開源項目給大家。現在終於又寫了一篇，是關於網絡方向的內容，希望對部分讀者有幫助。 1 WinDivert介紹 WinDivert是windows下爲數不多的非常優秀網絡庫，非常適合用於開發

jiulang 2022-10-18 14:29:51

ULID規範解讀與實現原理

前提 最近發現各個頻道推薦了很多ULID相關文章，這裏對ULID的規範文件進行解讀，並且基於Java語言自行實現ULID，通過此實現過程展示ULID的底層原理。 ULID出現的背景 ULID全稱是Universally Unique Le

throwable 2022-10-18 14:29:21

docker部署nacos配置mysql

version: "3" services: mysql: restart: always image: mysql:latest

Minifilter 是一種文件過濾驅動，該驅動簡稱爲微過濾驅動，相對於傳統的sfilter文件過濾驅動來說，微過濾驅動編寫時更簡單，其不需要考慮底層RIP如何派發且無需要考慮兼容性問題，微過濾驅動使用過濾管理器FilterManager提

yshark 2022-10-18 14:25:31

Linux 下搭建 Hadoop 環境

Linux 下搭建 Hadoop 環境 作者：Grey 原文地址： 博客園：Linux 下搭建 Hadoop 環境 CSDN：Linux 下搭建 Hadoop 環境 環境要求 操作系統：CentOS 7 下載地址 安裝說明 需要準備兩個節點

Grey Zeng 2022-10-18 14:25:11

Linux 下搭建 Kafka 環境

Linux 下搭建 Kafka 環境 作者：Grey 原文地址： 博客園：Linux 下搭建 Kafka 環境 CSDN：Linux 下搭建 Kafka 環境 環境要求 操作系統：CentOS 7 下載地址 安裝說明 Kafka 版本：2.

Grey Zeng 2022-10-18 14:25:11

Linux 下配置 hosts 並設置免密登錄

Linux 下配置 hosts 並設置免密登錄 作者：Grey 原文地址： 博客園：Linux 下配置 hosts 並設置免密登錄 CSDN：Linux 下配置 hosts 並設置免密登錄 說明 實現 Linux 下（基於 CentOS 7

Grey Zeng 2022-10-18 14:25:11

Net 逆向神器 dnSpy

下載地址:dnspy_

登峯人 2022-10-18 14:25:01

Redis 數據類型以及使用場景分別是什麼？

Redis 提供了豐富的數據類型，常見的有五種數據類型：String（字符串），Hash（哈希），List（列表），Set（集合）、Zset（有序集合）。 隨着 Redis 版本的更新，後面又支持了四種數據類型： BitMap（2.2

登峯人 2022-10-18 14:25:01

查看sessionid

-

登峯人 2022-10-18 14:25:01

爲什麼用 Redis 作爲 MySQL 的緩存？

主要是因爲 Redis 具備「高性能」和「高併發」兩種特性。 1、Redis 具備高性能 假如用戶第一次訪問 MySQL 中的某些數據。這個過程會比較慢，因爲是從硬盘上讀取的。將該用戶訪問的數據緩存在 Redis 中，這樣下一次再訪問這些數

登峯人 2022-10-18 14:25:01