

【贪吃蛇之二龙戏珠】

概要设计说明书

文档拟稿：【张松云】

建立日期：2017 年 7 月

文档审核：【审核人员】

审核日期：

文档页数：

当前版本：1.0

讯飞教育

更新记录

日期	更新人	版本	备注
2017.07.11	张松云	1.0	初稿完成

审核记录

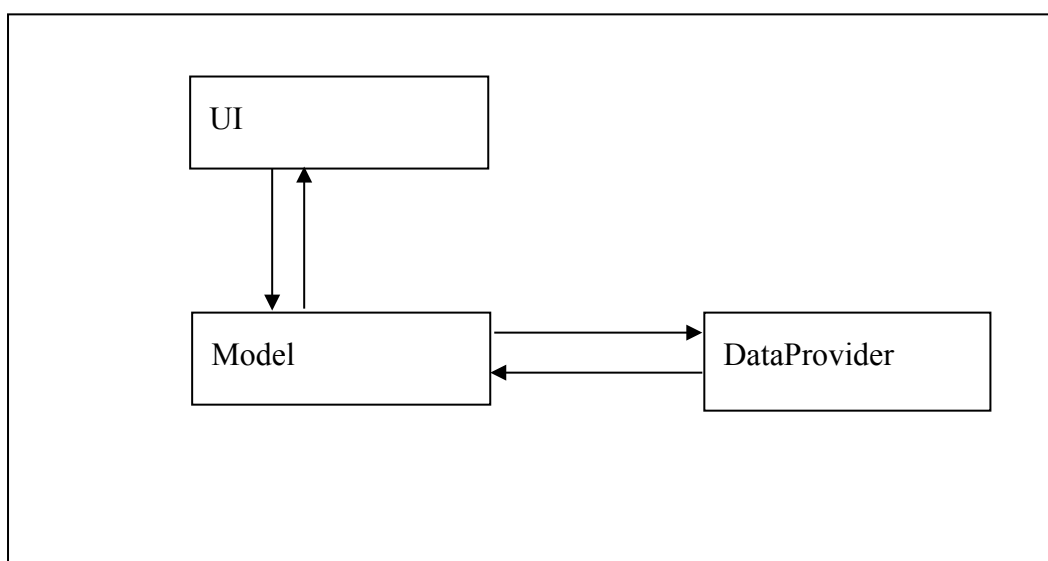
日期	审核人	职务	备注

1. 概述	2
2. 系统逻辑结构图	3
3. 代码处理逻辑	3
4. 关键技术	5
4.1 数据持久化	5
4.2 动态创建控件	5
4.3 多线程	5
4.4 画图	11

1. 概述

本文档用于说明贪吃蛇之二龙戏珠项目的概要设计，旨在帮助开发人员了解系统的整体架构，关键技术和关键模块的设计，帮助开发人员了解系统的设计概况，为详细设计做铺垫。

2. 系统逻辑结构图



比如：以上图的 UI 页面命名,创建普通用户页面，命名为：FrmUser.h

用户列表页面，命名为：FrmUserList.h

用户修改页面，命名为：FrmUserModify.h

控件，如登录按钮，命名为：btnLogin

用户名输入框，命名为：txtUserName

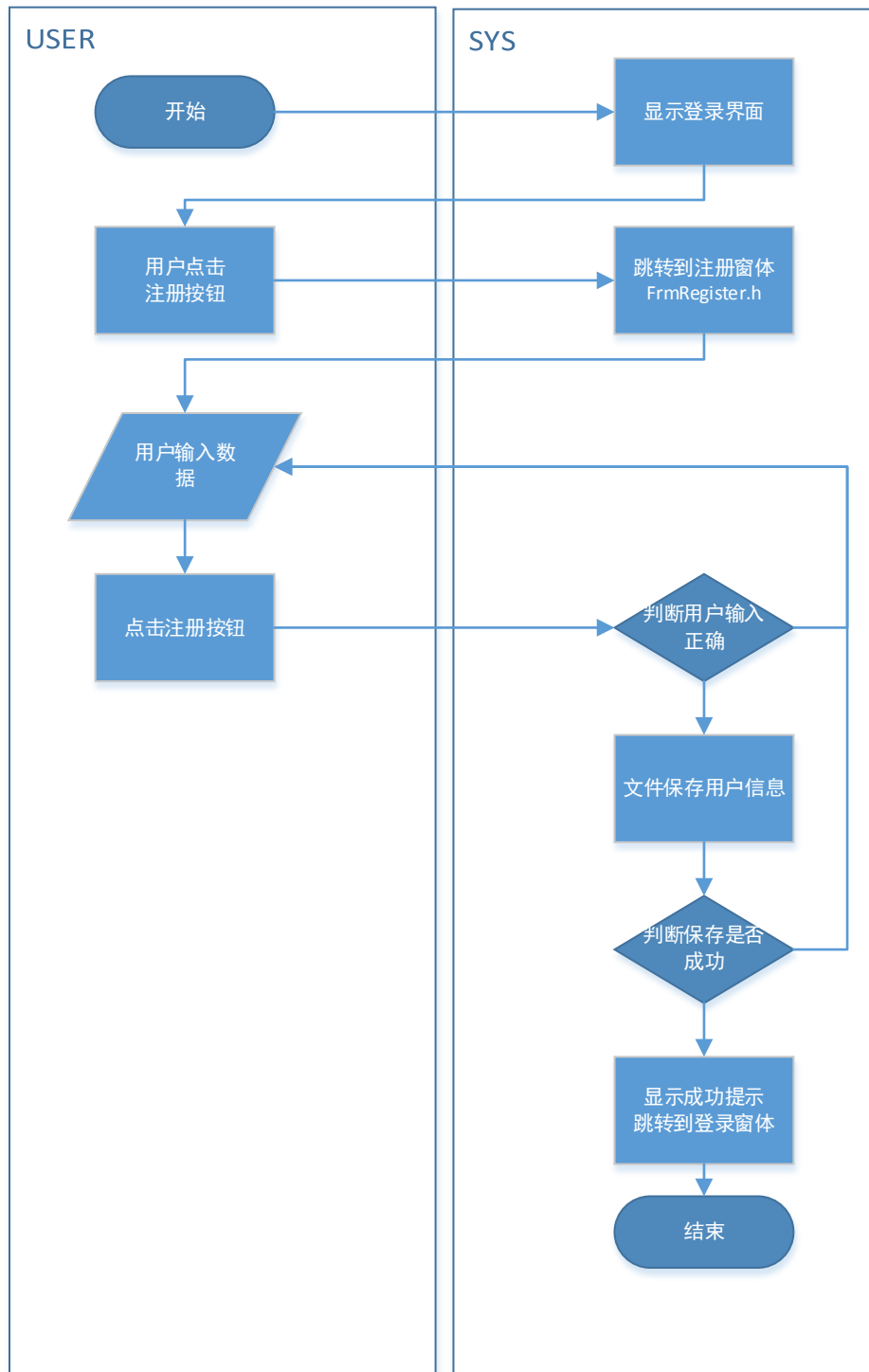
其他控件 Label，命名为：lblXXXX

两个单词的控件取首字母小写如 ListView，命名为：lvXXXX、ImageView，命名为：ivXXX

而 Model 命名，例如，用户，可命名为：User.h

3. 代码处理逻辑

本项目代码分层为 UI 层和 Model 层，UI 层负责数据的页面显示并处理数据并进行相关业务逻辑处理，然后进行页面跳转。以注册为例，整个逻辑处理流程如下图：



4. 关键技术

以下面列出一些常用关键技术的实现方式供参考。

4.1 数据持久化

见视频

4.2 动态创建控件

见视频

4.3 多线程

在编写大型程序时，将所有任务都放在一条线程上来完成并不是一个明智的选择，因为程序一旦卡死在某个环节或是发生了死循环，整个程序都会废掉。所以适当的用多线程有利于程序的良好运行。

在C++/CLR中使用多线程是很方便的，以下介绍使用方法：

- 添加命名空间：`using namespace System::Threading;`
- 创建新的线程：`Thread ^ oThread = gcnew Thread(gcnew ThreadStart(this, &data::df));`
- 启动新的线程：`oThread->Start();`

- 关闭新的线程：`oThread->Abort();` 在调用此方法的线程上引发

`ThreadAbortException`，以开始终止此线程的过程。调用此方法通常会终止线程。

- 挂起线程：`Thread::Sleep(n);` `n` 代表需要挂起的时间，单位为毫秒。哪个线程调用这个方法，哪个线程就挂起 `n` 毫秒。

- **This**: 指当前的窗口句柄或指针。
- **&data::df**: 你要在新进程里完成的任务。**df** 是 **data** 类中的一个函数。

代码示例:

```
// [C++]
//我用的编译环境是 vs2010
//使用 /clr 选项编译.
using namespace System;
using namespace System::Threading;

// 简单的多线程示例:在第二个线程中运行
//一个公有的方法 ThreadProc.
public ref class ThreadExample
{
public:

    //在新线程中调用 ThreadProc 方法。
    //这个方法将会运行十次，每次都会向控制台输出一行文本，然后
    //挂起很短的一段时间。运行十次后结束运行。
    static void ThreadProc()
    {
        for ( int i = 0; i < 10; i++ )
        {
            Console::Write( "ThreadProc: " );
            Console::WriteLine( i );

            // 把第二个线程挂起一段时间
            Thread::Sleep( 0 );
        }
    }
};

int main()
{
    Console::WriteLine( "主线程: 开启了第二个线程。" );
    //要说明的一点是，第一个线程就是当前 main() 函数所在的线程。

    // 创建一个线程，通过一个 ThreadStart 代理
    // 去代表 ThreadExample::ThreadProc 方法.
```

```
// 如果代理的是一个静态方法, 则不需要对象, 反之, 则需要调用对象的方法
//创建第二个线程。
Thread^ oThread = gcnew Thread( gcnew
ThreadStart(&ThreadExample::ThreadProc ) );

// 启动 ThreadProc 线程. 注意在单核电脑中, 新线程
// 要等主线程让出才能获取处理器
// 取消 oThread->Start(); 下面的 Thread::Sleep( 0 ); 的注释, 看看有什么不同
//启动第二个线程
oThread->Start();

for ( int i = 0; i < 4; i++ )
{
    Console::WriteLine( "主线程: 做了些事情。" );
    //下面这个 Thread::Sleep( 0 ); 的作用是把主线程挂起一段时间
    //Thread::Sleep( 0 );

}
Console::WriteLine( "主线程: 调用 Join(), 等待 ThreadProc 方法运行结束." );
oThread->Join();
Console::WriteLine( "主线程: ThreadProc->Join() 已经返回控制权. 按任意键
退出程序。" );
Console::ReadLine();
return 0;
}
```

上面的示例是控制台程序,但是在窗体程序中,我们开启子线程去完成一些耗时的工作,任务结束后需要更新控件,这个时候就不能直接在子线程中更新控件了,需要用到

SynchronizationContext 对象的Post方法.

示例如下:

下文中的&MyGame2::Form1::ChangeLocation, 表示的是 MyGame2 命名空间, Form1 窗体名, ChangeLocation 函数名

在类中创建一个异步上下文对象,在窗体类的构造中实例化

SynchronizationContext^ m_SyncContext ;


```
Form1(void)

{

    InitializeComponent();

    m_SyncContext = SynchronizationContext::Current;

}
```

创建在子线程中执行的方法

```
void ThreadChangePos()

{

    for(int i =0;i<1000;i++)

    {

        //上下文发送通知给一个代理对象SendOrPostCallback, 让这个对象去

        执行主线程更新ui的方法, 没有参数就写nullptr

        m_SyncContext->Post(gcnw

SendOrPostCallback(this,&MyGame2::Form1::ChangeLocation), nullptr);

        Thread::Sleep(100);

    }

}
```

主线程中更新ui的方法, 通过这个方法更新ui, 参数必须要写, 没有就声明 *Object*

```
void ChangeLocation(Object^ o)

{

    this->pictureBox1->Location = System::Drawing::Point(this->
```

```
>pictureBox1->Location.X, this->pictureBox1->Location.Y+5);  
  
}
```

最后在某个事件中去创建一个线程,并启动线程

```
Thread th = gcnew Thread(gcnew  
ThreadStart(this, &MyGame2::Form1::ThreadChangePos));  
  
th->Start();
```

```

1  #pragma once
2
3  namespace MyGame2 {
4
5      using namespace System;
6      using namespace System::ComponentModel;
7      using namespace System::Collections;
8      using namespace System::Windows::Forms;
9      using namespace System::Data;
10     using namespace System::Drawing;
11     using namespace System::Threading;
12
13     /// <summary>
14     /// Form1 摘要
15     /// </summary>
16     public ref class Form1 : public System::Windows::Forms::Form
17     {
18     public:
19         //创建上下文对象
20         SynchronizationContext^ m_SyncContext ;
21         delegate void ChangeDelegate();
22         Form1(void)
23         {
24             InitializeComponent();
25             //
26             //TODO: 在此处添加构造函数代码
27             //实例化
28             m_SyncContext = SynchronizationContext::Current;
29         }
30         //主线程更新ui的函数
31         void ChangeLocation(Object^ o)
32         {
33             this->pictureBox1->Location = System::Drawing::Point(this->pictureBox1->Location.X,
34                 this->pictureBox1->Location.Y+5);
35         }
36         //在子线程中执行的方法,在子线程中通知主线的更新函数
37         void ThreadChangePos()
38         {
39             for(int i =0;i<1000;i++)
40             {
41                 //通知主线程更新
42                 m_SyncContext->Post(gcnew SendOrPostCallback(this,&MyGame2::Form1::ChangeLocation),nullptr);
43                 Thread::Sleep(100);
44             }
45         }
46     protected:
47         /// ...
48         Form1() { ... }
49     private: System::Windows::Forms::PictureBox^ pictureBox1;
50     protected:
51     private: System::Windows::Forms::Button^ button1;
52     private: System::ComponentModel::IContainer^ components;
53     private:
54         /// <summary>
55         /// 必需的设计器变量。
56         /// </summary>
57         Windows Form Designer generated code
58     private: System::Void button1_Click(System::Object^ sender, System::EventArgs^ e) {
59         Thread th = gcnew Thread(gcnew ThreadStart(this,&MyGame2::Form1::ThreadChangePos));
60         th->Start();
61     }
62 };
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

```

4.4 画图

绘制电脑图片到窗体指定位置

```
Image^ img = Image::FromFile("c:/gl.jpg");//建立Image对象  
Graphics^ g = Graphics::FromImage(img);//创建Graphics对象  
  
int width = 300;  
  
int height =200;  
  
e->Graphics->DrawImage(img, PointF(10, 10));
```

详见官方文档:

[https://msdn.microsoft.com/zh-cn/library/system.drawing.graphics_methods\(v=vs.110\).aspx](https://msdn.microsoft.com/zh-cn/library/system.drawing.graphics_methods(v=vs.110).aspx)

双缓冲技术

双缓冲是将图片在显示到 DC 前,现在要内存建一个 DC,也就是用于存储这张图片的内存区,然后在将这部分 update 到你要显示的地方

这样,可以防止画面抖动很大

这样和你说吧,如果要实现你要的效果,你必须用指针访问内存

比如,把程序声明成 unsafe 的,然后按照上面的操作进行

```
BufferedGraphicsContext^ current = BufferedGraphicsManager::Current; //(1)
```

```
BufferedGraphics^ bg = current->Allocate(this->CreateGraphics(),this->DisplayRectangle); //(2)
```

```
Graphics^ g = bg->Graphics; //(3)
```

```
//随机 宽 400 高 400
```

```
Random^ rnd = gcnew Random();
```

```
int x,y,w,h,r,i;
```

```
for (i = 0; i < 10000; i++)
```

```
{
```

```
    x = rnd->Next(400);
```

```
    y = rnd->Next(400);
```

```
    r = rnd->Next(20);
```

```
    w = rnd->Next(10);
```

```
    h = rnd->Next(10);
```

```
    g->DrawEllipse(Pens::Blue, x, y, w, h);
```

```
}
```

```
bg->Render();//(4)
```