# LTCingFW框架说明

## 一、概述

LTCingFW是一个轻量级C#Winform应用的框架，它主要用于帮助程序简单化编程和整体规范化架构。是一个由学习、思考、创造而来的过程所造就的框架，在这个过程中，本着一切简单化的原则，尽可能的让一切可读性、操作性提高，让程序员的工作量变少。

## 二、IOC

IOC(Inversion of Control)控制反转，用于减少程序间的耦合度。我们将程序的模块组件的依赖关系交给框架来管理，而不是由我们自己创建，自己把一些实例赋值给另一些实例中，这会导致各个组件的依赖十分混乱。

在框架中，由框架来管理我们组件的生命周期，包括框架启动时创建和程序结束时释放销毁，在通过依赖注入的方式，使得各个组件能够得到其他组件的引用。

在这个过程中，我们需要完成以下两个步骤：

1. 注册组件实例
2. 实例注入

##### 2.1注册组件实例

注册组件实例有以下两种方法：

1. 使用特性

图2-1

图2-1是框架中使用的特性的继承关系，在实际中我们需要使用[Controller][Service][Dao]来告诉框架我们注册的实例是什么。这3种特性是针对类的特性，框架会将标注了以上特性的类自动生成实例。这些实例作为Bean存储于我们框架的LTCingFwSet中的Beans属性中，我们可以从中按照Name属性提取出来。

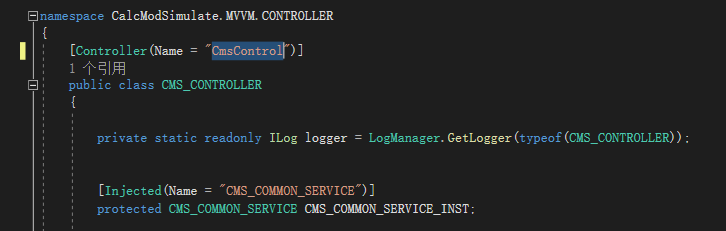


图2-2

图2-2中的[Controller]表示该类会被框架自动创建实例。它是一个Controller组件。该组件的名字为CmsControl。当我们不配置特性的Name属性时，默认会把类名作为Name属性的值。

1. 在配置文件中配置

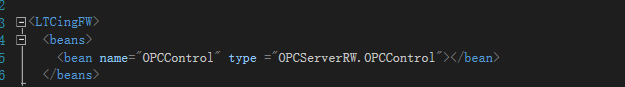


图2-3

对于程序外部引用的类，我们可以在LTCingFW.xml配置文件中，配置组件类的名字和全名。

##### 2.2 实例注入

所有的组件实例都是已经创建好的，获取实例的引用，有两种方式：

1. 使用【inject】特性

在图2-2中，我们看到有一个名为CMS\_COMMON\_SERVICE的Service被注入给了Controller，此时我们没有创建过该Service，便可以使用引用的Service了。

1. 手动提取

实例注入完全是在组件实例之间完成的，当我们想在一个并不是组件的类（比如一个静态类）中使用组件实例，我们需要无法使用注入的方式，但是可以从框架中提取出该组件实例。方法如下：

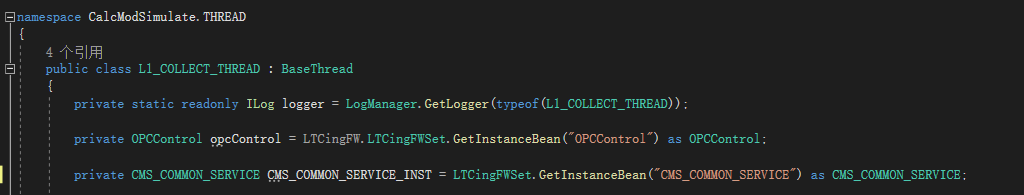


图2-4

图2-4中在线程中得到名为OPCControl的组件实例的引用和名为CMS\_COMMON\_SERVICE的组件实例的引用。

## 三、MVC架构

##### 整体架构

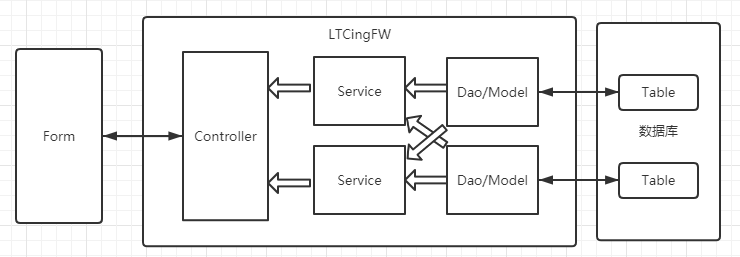


图3-1

图3-1中的空心箭头表示依赖关系，实心箭头表示数据交互。

依赖关系请按照图中所示，不允许出现Controller依赖Controller的情况，即高级依赖低级的关系。

##### Dao数据连接层

Dao意思是Data Access Object，它的作用就是和数据库交互，一个Dao组件实例对应一个数据库中的一个表，一个Dao组件实例中的方法便是对一个数据库中的一个表的增删改查操作。它的可操作范围空间仅仅为一个数据库中的一个表。

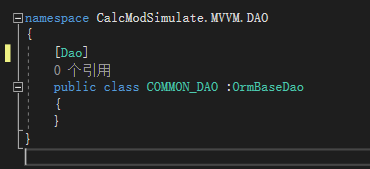


图3-2

DAO层使用【DAO】特性标注，框架已经给出了基础的Dao类(名为OrmBaseDao)，并将该类作为组件实例注册到了框架中，该方法提供几乎所有用到的增删改查方法（条数查询系、分页查询系、查询系、新增系、修改系、删除系），继承该类即可使用其所有的方法。

在LTCingFW框架中，DAO组件仅仅是一个方法的集合，不包含任何表的信息，诸如表名、列名、列类型等信息全部来自OrmBaseModel类。即相比于传统MVC框架，LTCingFW的DAO层不包含任何特殊的信息，因此一个公共的DAO组件实例即可满足所有的Service组件实例。

或者我们可以不写DAO层，直接把OrmBaseDao注入到我们的Service组件实例中去。或者不使用DAO层，因为OrmBaseModel也继承了OrmBaseDao，所以可以如下写：

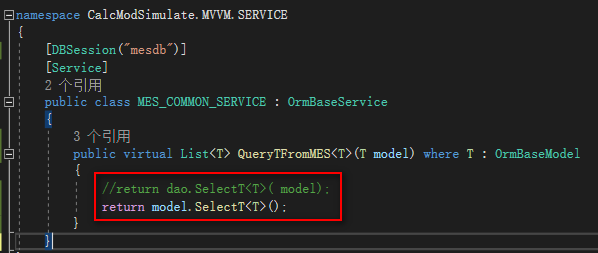


图3-3

图3-3中的两句话是一样的，所以可以直接省略DAO层。

##### Service服务层

Service服务层内可以注入多个Dao组件实例，因此服务层的方法可以对同一个数据库的多个进行操作，服务层的每一个方法给与一个数据库连接（DBSession），告知该服务层方法需要连接哪个数据库，对于非查询操作可以在DBSession中开启事务，以确保多表操作的原子性。

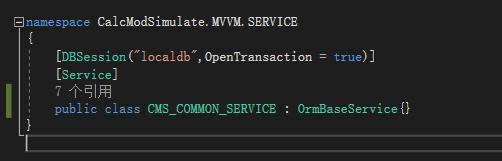


图3-4

Service层使用[Service]特性标注，框架给出了基础的Service类(名为OrmBaseService),该类包含简单的单表增删改查方法。

在使用该类的时候需要注意：

1，原则上我们希望一个Service只对应一个数据库，可操作范围是一个数据库。所以我们在自己的定义的Service类上加入【DBSession】特性，则该类的所有方法都使用相同的的数据库连接。

2，OrmBaseService类已经有注入OrmBaseDao，所以如果我们的Service类继承了OrmBaseServcie，可以不写注入DAO。

3，OrmBaseService类中有对于单表操作的简单方法，基本上可DAO层的方法一样，这些方法的DBSession全部都是使用Service类上标注的【DBSession】特性来建立数据库连接，所以如果Service类上没有标注【DBSession】特性，这些方法会找不到DBSession而报错。多表操作还需要自己写方法。

Service层的每个方法都可以添加自己的【DBSession】特性，但是我们建议这个方法写到另建的Service类中去。如下图：

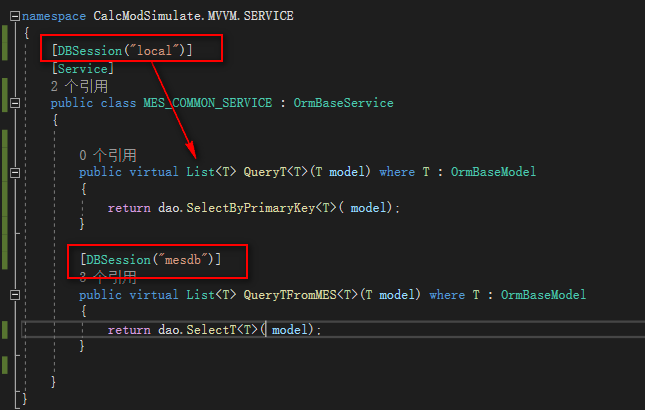


图3-5

在方法的DBSession优先级要大于在类上的DBSession。请注意所有的Service层的方法全部使用virtual 方法，表示该方法是虚方法，因为框架会对这些方法进行重写，如果不加virtual，该方法便无法得到DBSession数据库连接。

##### Controller控制层

Controller层中可以注入多个Service组件实例，该层的方法可以操作多个Service，即多个数据库，也就是最大的可操作范围。是主要的业务逻辑编写地点。在该层可以掌握程序的全局信息来处理业务。

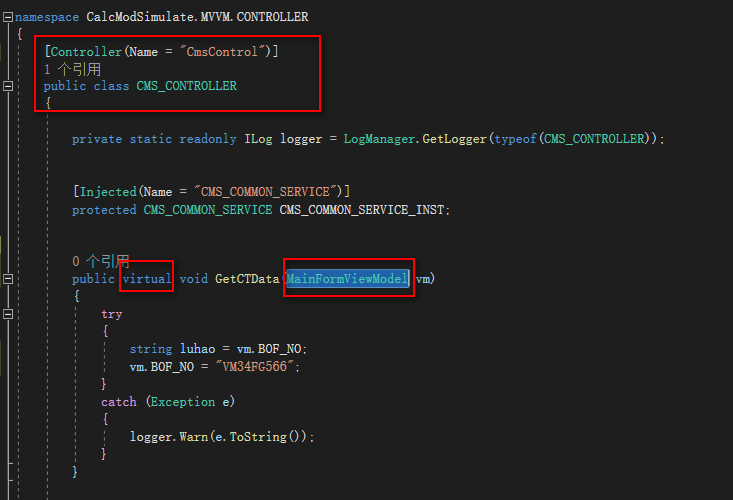


图3-6

Controller层使用【Controller】特性标注。Controller的所有方法必须使用virtual，图3-5中GetCTData方法的参数是MainForm的双向绑定实例。该实例中的属性和MainForm中的文本框等组件的内容双向绑定，即改变一方，另一方会立马改变。图中的方法读取BOF\_NO属性，然后又给BOF\_NO赋值，改变了页面上的显示。之后会将双向绑定的方法。

## ORM

ORM，即对象关系映射（英语：Object Relational Mapping，简称ORM，或O/RM，或O/R mapping），其主要的目的是建立程序的类和数据库中的表的对应关系，即一个ORM类就是一个数据库表。数据库中的每一行数据对应程序中ORM类（在本框架中是继承了OrmBaseModel的类）的一个实例，表的列是实例的属性。在编写我们的ORM类的过程中，即是编写对应数据库中表的信息。

LTCingFW框架中使用的ORM类如下图所示：

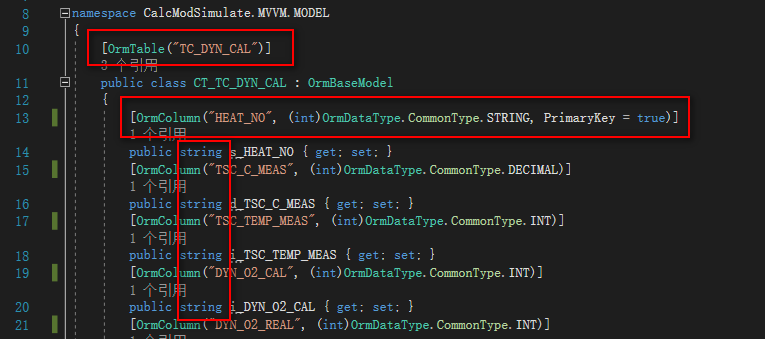


图4-1

1. 类必须以【OrmTable】进行标注，其后必须跟着实际数据库中的实际表名。
2. 类必须继承OrmBaseModel。
3. 类中属性，凡是用于对应表中的列，则必须使用属性。
4. 类中属性，凡是用于对应表中的列，则必须标注【OrmColumn】特性，三个参数分别是列名、列类型、是否为主键。列类型分为六种，在OrmDataType.CommonType枚举中选择：字符、小数、整形、布尔、日期。框架会根据数据类型进行读取、写入时的类型转换。
5. 类中属性，凡是用于对应表中的列，则必须使用string类型。框架会将结果转为string类型。如果用于计算等，请使用时自行转换。
6. 类中的列对应属性，是真实数据表中列集的子集，但是主键列必须包含，因为如果主键列不全，进行的数据库操作可能失败。

## DBSession

DBSession是数据库连接的意思，我们需要配置好连接信息。

##### 5.1 配置

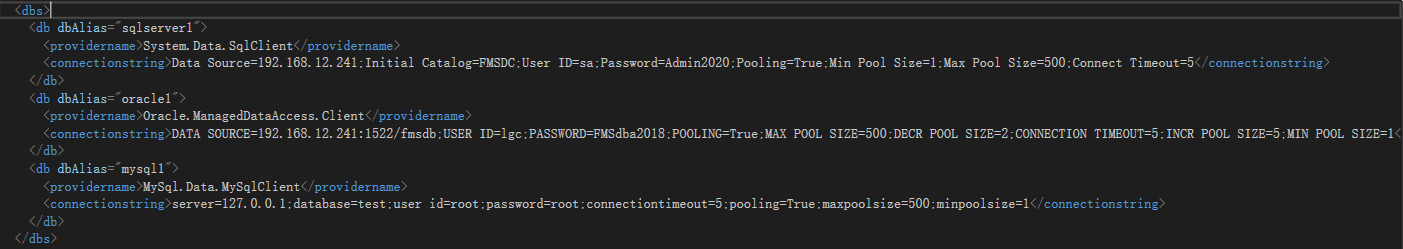


图5-1

1. DBSession的配置写于LTCingFW.xml中的<dbs></dbs>中。
2. dbAlias为别名，自己定义好后，作为识别标识使用。

3. Providename表示其连接为mysql、oracle、sqlserver，且为固定值，分别为【MySql.Data.MySqlClient】【Oracle.ManagedDataAccess.Client】【System.Data.SqlClient】，mysql使用5.5到5.7版本，其他版本未测试，oracle使用9i以后版本，sqlserver使用2008以后版本。

4. Connectingstring是连接字符串，由数据库供应商自行定义的写法，可以从网上找到。

##### 5.2 使用

从MVC架构来看，Service层的方法对应了一个数据库范围的操作，所以DBSession主要用于Service层的方法上。可以写在Service的类上或者方法上。需要注意Service层的方法都必须是virtual方法，如图3-4。

如果Controller层的方法仅仅使用一个数据库，也可以将DBSession写在Controller层的方法上。此时Controller层调用内部Service层的方法时，使用的是外层Controller方法的DBSession，也就是说外层DBSession优先级大于内层DBSession优先级，这样可以减少DBSession的创建。但是一般无需在Controller层上添加DBSession。

另外，DBSession可以使用如下方式从上下文中提取出来，如图5-2：

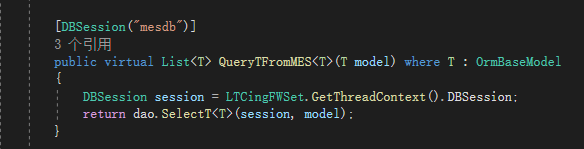


图5-2

可以使用LTCingFWSet.GetThreadContext().DBSession获取。

##### 5.3 事务

可以为DBSession开启事务，如图5-3所示：

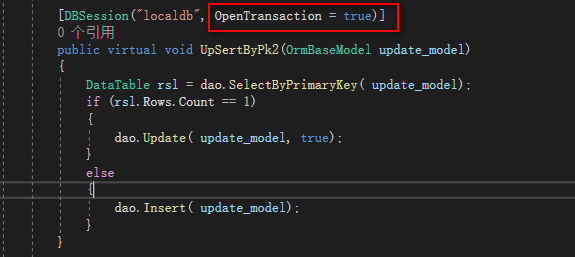


图5-3

开启的语法是，为DBsession特性添加OpenTransaction = true，开启事务后Service层的方法内，对多表的多次操作会使用同一事务，若中间有操作失败会一起回退。一般只针对于包含有新增、修改、删除的操作。

##### 5.5动态注册

DBSession是存放于LTCingFWSet中的，可以动态添加，语法如图5-4所示：

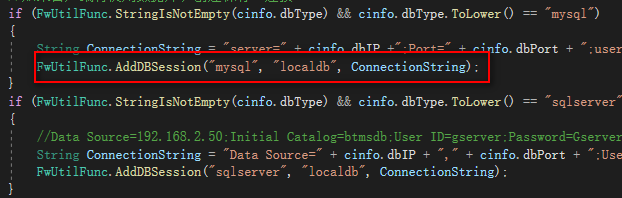


图5-4

使用FwUtilFunc.AddDBSession方法，三个参数分别为，数据库类型(mysql,sqlserver,oracle),别名，连接字符串。

当我们确保已经注册之后，可以使用DBSession.OpenSession(别名，是否开启事务)的方式，创建DBSession。

## AOP

AOP是面向切面的意思，当我们想针对某一类方法添加统一的操作时，可以将这一类的方法看做一个切面，在方法的执行前后添加一些操作。

##### 配置

使用AOP需要配置在LTCingFW.xml中，如图6-1所示：

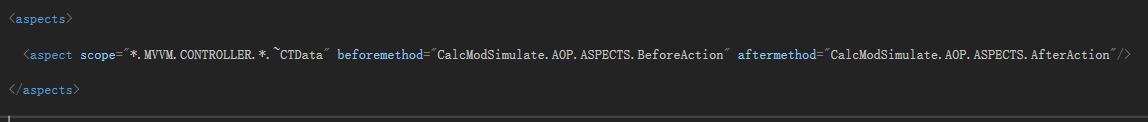


图6-1

方法配置需要填写几个参数：

scope：表示针对于哪些方法，【\*】表示任意路径名，【.】表示路径间隔，【~】表示任意不完整字符串，如~CTData表示以CTData结尾的所有字符串。

beforemethod：表示在切面方法之前执行的方法的路径和方法名，路径和方法名请对应参考图6-2。

aftermethod：表示在切面方法之后执行的方法的路径和方法名，路径和方法名请对应参考图6-2，即便原方法出现错误，此方法也会执行。

##### 切面方法

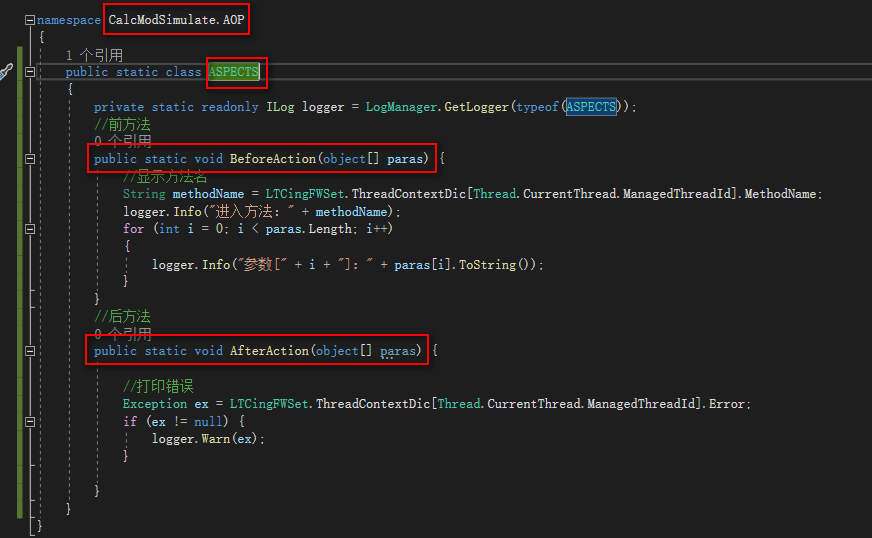


图6-2

框架对切面方法有要求：

1. 切面方法必须为公共的静态函数。
2. 切面方法的参数必须为object[],框架会将原函数的参数作为数组传入。
3. 错误提取，可以通过以下方式从切面方法中提取原方法发生的错误：

LTCingFWSet.ThreadContextDic[Thread.CurrentThread.ManagedThreadId].Error

1. 如果有多个切面匹配到方法，只用第一个切面。

## VIEWMODEL

ViewModel的作用是与Winform页面上的诸如文本框等的控件显示内容做双向绑定，即改变一方，另一方自动作出改变。使得页面的内容和ViewModel中的属性一一对应。如果我们在Controller中想要改变页面内容，直接修改ViewModel参数即可，解耦了业务逻辑程序和页面显示之间的紧密关系，规避原有的将逻辑写于Winform页面中导致庞大复杂臃肿的局面。

编写ViewModel需要以下几个步骤：

##### 7.1 编写ViewModel

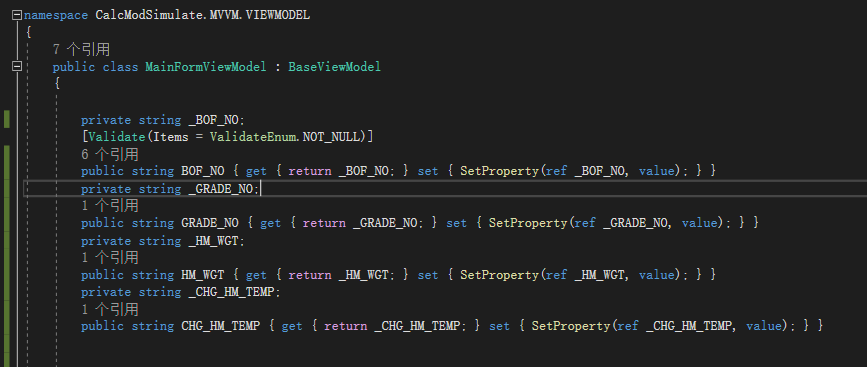


图7-1

1. ViewModel类请继承BaseViewModel类
2. 创建私有变量和对应属性，get方法和set方法按照图示中书写，其中SetProperty为基类BaseViewModel中的方法。

##### 7.2 在Winform中绑定控件属性

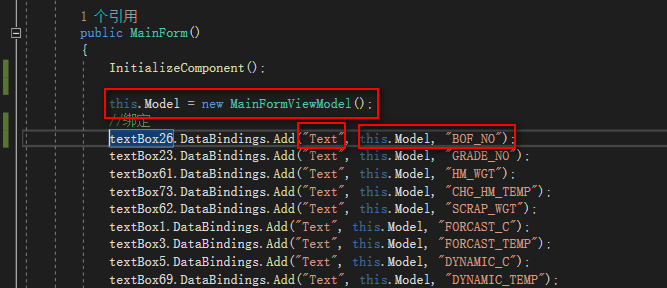


图7-2

1. 创建ViewModel实例。
2. 将控件的属性绑定到ViewModel实例的属性上，语法为：

控件.DataBindings.Add(控件属性名，ViewModel实例，ViewModel实例的属性名)

如图中案例：将TextBox的Text的属性与MainFormViewModel的BOF\_NO属性绑定。

##### 7.3 属性有效性验证

框架可以为ViewModel的绑定属性做有效性验证，参见图7-1。实现验证需要在ViewModel的属性上（注意非字段）添加【Validate】特性，该特性有两个验证属性，分别是Item验证和Regx验证，Item验证包括非空、最大字符串长度、最小字符串长度、是否为日期、是否为数字；Regx验证是正则表达式验证，可以同时使用两种方式验证。

1. Item验证：选择LTCingFW.ValidateEnum中的一种，分别是：

NOT\_NULL（非空），不可为空。

MAX\_LENGTH（最大长度），不可超过最大长度，使用该验证给【Validate】特性的MaxLength属性赋值。

MIN\_LENGTH（最小长度），不可少于最小长度，使用该验证给【Validate】特性的MinLength属性赋值。

IS\_DATE （日期），必须为日期的字符串，即使Convert.ToDateTime(value)无错即可。

IS\_NUMBER（数字），必须为数字的字符串。

1. Regx验证：给【Validate】特性的Regx属性写入正则表达式后，自动启用。框架自动验证验证属性的值是否与【Validate】特性的Regx正则表达式相匹配（IsMatch）。

验证结果存放于ViewModel实例的ValidResultDic字典里，以属性名为key，以ValidResult为值。ValidResult里有验证结果、错误信息、属性名、属性值。可以使用ValidResultDic.clear()清空验证结果。

## 其他功能

##### 8.1 Winform命令异步执行

框架自带Winform增强功能，使用需要使Winform继承框架的BaseForm。通常我们点击按钮后如果运行过长，会导致界面卡死，如果使用异步执行功能，则不会出现该情况。

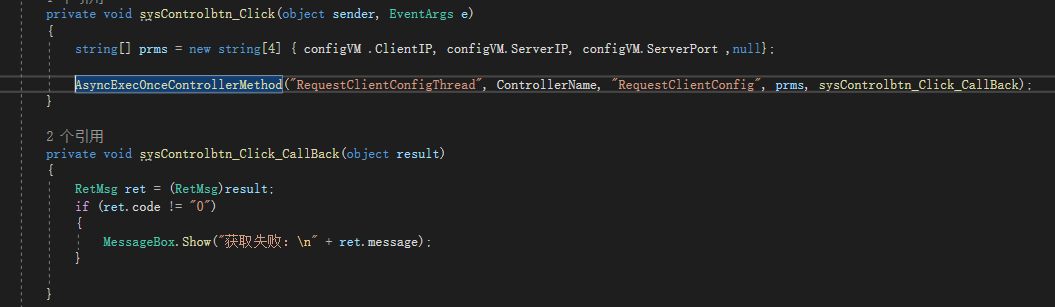


图8-1

使用BaseForm中的AsyncExecOnceControllerMethod方法，第一个参数是线程名，第二个参数是【Controller】组件实例的名字，第三个参数是Controller里的执行方法名，第四个参数是执行方法的参数集和（object [ ]），第五个参数是回调函数，回调函数的参数必须为object，返回值为void。

##### 8.2 简单线程

框架自带了简单线程的基类BaseThread和简单线程的启停方法。减少编写线程时的代码量。使用简单线程需要自定义线程类，并继承BaseThread。继承之后自定义实现run方法。

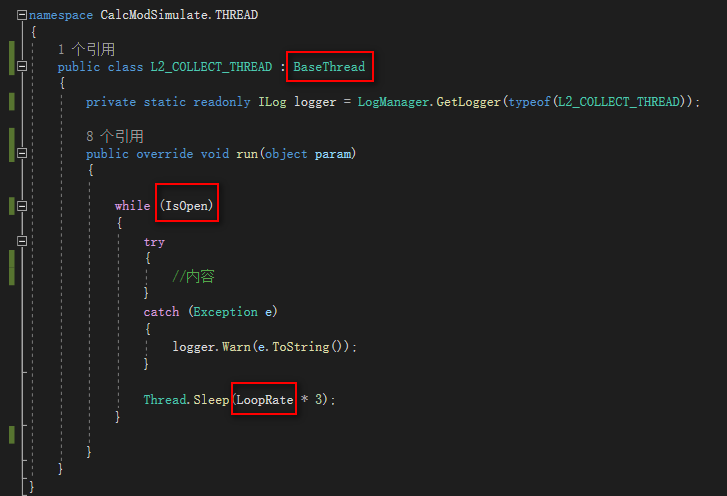


图8-2

图8-2给出了简单线程类的基础写法，该线程类继承了BaseThread，并实现了BaseThread类的抽象方法run，IsOpen是BaseForm中的属性，关闭线程会将IsOpen置为False，线程自动结束。LoopRate是BaseForm的属性，表示循环间隔，默认为1000ms，可以自行修改。

简单线程的启停分别使用以下方法：

启动：FwUtilFunc.OpenThread(threadContext,threadName,param);

参数1：继承了BaseThread的线程类实例，参数2：线程名，参数3：run方法的参数。

停止：FwUtilFunc.CloseThread(threadName);

参数1：线程名。

##### 8.3 HTTP请求

框架自带了简单的HTTP请求方法。包括GET方法和POST方法。

GET方法：

LTCingFW.utils.HttpUtil.HttpGet(string Url)

参数1：访问地址

返回结果字符串。

POST方法：

LTCingFW.utils.HttpUtil.HttpPost(string url, string postDataStr,int waitTime\_ms)

参数1：访问地址，参数2：访问时附带内容，参数3：等待时间

返回结果字符串。