## C#读书笔记

### 数据类型

static readonly ,const

static readonly :static readonly是在运行时计算出其值的

const:const的值是在编译期间确定的,编译后值直接存入dll

Public class P

{

const int A=B\*10;

const int B=10;

public static void Main(string[] args)

{

Console.WriteLine("A is {0},B is {1} ",A,B);

}

}

### 类

### 继承

见用户自定义异常。

### 委托

委托首先解决的是方法作为参数调用的问题

1.委托的回调：如果想把方法作为参数传递，就定义一个委托类型的形参

1. 委托链：执行多个方法

(此处思考多人订阅同一个事件)

1. 委托和事件结合
2. 把一个外部类的方法传递到类的内部，可以在类的内部定义一个委托事件

(这就是用委托不直接使用方法的原因，虽然调用方法f1(),f2()也能够调用方法)

5.把方法委托到线程执行

6.委托执行一个匿名方法即Lambda表达式

Lambda表达式实质是匿名方法，可以和代理结合，但是本身不是代理。如果参数是Action

New Action(()=>{...})，可以这样传匿名方法，还可以这样写Action a = () => { };。

最好是有名称的，此时不需要确定实参，目的是传递方法，实参在方法内部给定。

1. 委托的协变和逆变

### 事件

事件最主要的是事件的订阅者和事件的发布者，事件的发布者发布事件的功能。事件的订阅者通过委托链定义多个事件。其他对象也可以订阅。

public partial class Form1 : Form

{

**public EventHandler btn1;**

**public EventHandler btn2; // 可以提供给外部调用**

public Form1()

{

InitializeComponent();

}

private void Form1\_Load(object sender, EventArgs e)

{

**btn1 = new EventHandler(butt1);**

**btn2 = new EventHandler(butt2);**

**this.button3.Click += btn1;**

**this.button3.Click += btn2;**

**this.button4.Click += btn2;**

}

private void butt1(object sender, EventArgs e)

{

this.button1.Visible = false;

}

private void butt2(object sender, EventArgs e)

{

this.button2.Visible = false;

}

}

2. 见委托4的功能

### 线程

1. private void button2\_Click(object sender, EventArgs e)

{

//public delegate void run();

//public delegate void mydele(string s);

**mydele dele = new mydele(temp);**

**dele.BeginInvoke("开启线程异步计算...", null, null);**

// Delegate.BeginInvoke 会从线程池取一个线程异步执行,Delegate.InInvoke运行在当前线程上

}

public void temp(string s)

{

MessageBox.Show(s);

for (int i = 0; i < 2000; i++)

{

for (int j = 0; j < 100000; j++)

{

Count += j;

}

}

**run r = new run(fun);**

**this.BeginInvoke(r, null);**

MessageBox.Show("123");

// Control.BeginInvoke ,Control.InInvoke 都只是运行在UI线程上，

// Control.BeginInvoke 异步指的是MessageBox.Show("123")会BeginInvoke 后立刻执行

}

private void fun()

{

this.textBox3.Text = Count.ToString();

}

2. private void button2\_Click(object sender, EventArgs e)

{

//public delegate void mydele(string s);

**mydele dele = new mydele(temp);**

**dele.BeginInvoke("开启线程异步计算.....",fun,null);**

}

public void temp(string s)

{

MessageBox.Show(s);

for (int i = 0; i < 2000; i++)

{

for (int j = 0; j < 100000; j++)

{

Count += j;

}

}

}

private void fun(**IAsyncResult** ar)

{

**Action a = new Action(() =>**

**{**

**this.textBox3.Text = Count.ToString();**

**});**

**this.BeginInvoke(a, null);**

}

### 接口

### 泛型

### 流和文件

TeaxReader ，XmlReader 和 BinaryReader 格式的读写器

TeaxReader 是 StringRrader 和 StreamReader 的抽象父类

XmlReader 是 XmlTextReader 和 XmlNodeReader 的抽象父类

从两个子类的构造函数和方法看出StringRrader不能操作流（很奇怪）

其他几个格式的读写器可以

File FileInfo 能够提供 FileStream 流供读写器读写

### 用户自定义异常

1. 继承Exception 类
2. 在派生类中指定基类的构造函数。

class \_UserException : Exception

{

public string ex\_message = string.Empty;

public \_UserException(): base()

{

}

public \_UserException(string message)

: base(message)

{

this.ex\_message = "用户自定义异常：" + message;

}

}

public \_UserException(string message,OtherException ex):base(message,ex)

{

}

此处可以catch一段代码抓取OtherException ( 可以为 OtherException 1，OtherException 2 )，

然后统一throw UserException供外部抓取