1. Асинхронные вызовы

Для выполнения функции можно воспользоваться объектом делегата, инкапсулирующего указатель этой функции. Вызов же функции можно осуществить как синхронно, так и асинхронно.

При *синхронном вызове* выполнение программы, вызвавшей функцию, приостанавливается, и управление передаётся вызванной функции. После завершения работы этой функции вызвавшая программа возобновит работу с прерванной точки.

При *асинхронном вызове* инициируется выполнение вызванной функции в *другом* потоке. Поэтому выполнение вызвавшей программы и вызванной *асинхронной функции* осуществляется *параллельно* (в разных потоках).

Возможны два варианта асинхронного вызова функции: *вызов без функции обратного вызова* и *вызов с функцией обратного вызова*. Под *функцией обратного вызова* понимается функция – обработчик события, генерируемого при завершении выполнения асинхронной функции.

Компилятор при генерировании класса *асинхронного делегата*, то есть делегата, инкапсулирующего асинхронную функцию, добавляет в этот класс специальные функции BeginInvoke() и EndInvoke(), которые обеспечат асинхронный вызов.

Выполняя асинхронный вызов, функция BeginInvoke() возвращает объект типа AsyncResult, который может быть использован как для выявления того, завершился ли вызов, так и для получения результатов выполнения асинхронной функции.

Для ожидания завершения потока, выполняющего асинхронный вызов, в вызывающем потоке применяют функцию EndInvoke().

Функция обратного вызова является функцией, которая начинает выполняться сразу после окончания работы асинхронной функции.

Формат *функции BeginInvoke()* инициирования асинхронного вызова следующий:

```
public  virtual IAsyncResult    BeginInvoke (object[] args,
                                             AsyncCallback asyncBack_del,
```

Параметры асинхронной функции, объект делегата функции обратного вызова, объект асинхронного делегата.

Делегат обратного вызова:

```
public delegate void AsyncCallback (IAsyncResult ar  );
```

Функция обратного вызова имеет единственный параметр, являющийся возвращаемым объектом результата функции BeginInvoke(), то есть объектом типа AsyncResult, наследуемого *интерфейс IAsyncResult*.

```
public interface IAsyncResult
{
    Object AsyncState {get;}
    WaitHandle AsyncWaitHandle {get;}
```

```
    bool CompletedSynchronously {get;}
    bool IsCompleted {get;}
}
```

Свойство AsyncState возвращает объект состояния, указанный в качестве последнего аргумента функции BeginInvoke().

Свойство AsyncWaitHandle может быть использовано для синхронизации потока вызвавшей программы с потоком вызванной асинхронной функции.

Значение *свойства CompletedSynchronously*, равное true, указывает, что функция BeginInvoke выполнилась синхронно.

Свойство IsCompleted получит значение true после завершения асинхронного вызова.

Объект класса AsyncResult кроме перечисленных свойств интерфейса IAsyncResult включает дополнительные свойства AsyncDelegate и EndInvokeCalled.

Свойство AsyncDelegate возвращает объект делегата, делегирующего асинхронную функцию. А *свойство EndInvokeCalled* указывает, была ли вызвана функция EndInvokeCalled().

Функция EndInvoke() имеет следующий формат:

public virtual object **EndInvoke** (IAsyncResult *async-res*);

Параметр - объект, возвращаемый функцией BeginInvoke..

Алгоритм реализации асинхронного вызова

Объявляем делегат для асинхронной функции – асинхронный делегат, определяющий формат асинхронной функции со следующим порядком параметров в списке параметров функции:

- включить все in параметры,
- включить все out параметры,
- включить все in/out параметры,
- включить все ref параметры.

Затем *создаём объект этого асинхронного делегата*, инкапсулировав в нём указатель на асинхронную функцию.

Теперь, используя ссылку на созданный объект асинхронного делегата, *вызываем его функцию BeginInvoke()*, присвоив при этом её возвращаемое значение интерфейсной ссылке типа IAsyncResult. В начале списка аргументов функции BeginInvoke() помещаем аргументы функции асинхронного вызова в такой же последовательности, как определено асинхронным делегатом.

Если функция обратного вызова не используется, то первый и второй дополнительные аргументы функции BeginInvoke() делаем равными null.

Если же используется функция обратного вызова, то помещаем ещё два последних аргумента.

В предпоследнем (первом дополнительном) аргументе передаём объект делегата обратного вызова типа AsyncCallback, инкапсулировав в нём указатель на функцию обратного вызова. В качестве последнего (второго дополнительного) аргумента передаём объект асинхронного делегата, инкапсулирующего указатель асинхронной функции.

Для приостановки выполнения вызвавшей программы при отсутствии функции обратного вызова в требуемой точке вызвавшей функции необходимо воспользоваться функцией EndInvoke().

Если же применена функция обратного вызова, то функция EndInvoke(), вызванная в теле функции обратного вызова, позволит получить требуемые результаты.

При *вызове функции EndInvoke()* необходимо придерживаться следующего порядка указания аргументов в списке её аргументов:

- включить все in/out параметры,
- включить все out параметры,
- включить все ref параметры,
- включить в качестве последнего аргумента объект типа AsyncResult.

Также для функции EndInvoke() необходимо обеспечить возврат её значения, являющегося возвращаемым значением асинхронной функции.

Пример 1.1. Асинхронный вызов без использования функции обратного вызова.

```
///////////////
// C#  File  csK_Async_7
using System;
using System.Threading;

// Асинхронный делегат
delegate int Del (string s, out int nOut, ref int nRef);

class A
{
	// Асинхронная функция
	public int F (string s ,out int nOut, ref int nRef)
	{
		// Выдать три значения s в асинхронной функции
		int n= 0;
		while (n < 3)
		{
			Console.Write (s);
			Thread.Sleep (10);
			n++;
		}
		nOut= n; nRef= n;
		return n;
	}
}

class Test
{
	// Основная функция Main(), вызвавшая асинхронную функцию F
```

```
        // без функции обратного вызова
        static void Main (string[] args)
        {
                A sA= new A ( );

                // Получить объект делегата для асинхронного вызова функции F()
                Del del= new Del (sA.F);

                // Отметить начало выполнения вызвавшей функции Main()
                Console.Write ("a");
                int nOut= 0;
                int nRef= 0;

                // Инициировать вызов асинхронной функции F
                IAsyncResult asyncResult= del.BeginInvoke ("b",
                                                out  nOut,  ref nRef, null, null);
                // int n= del.EndInvoke (out nOut, ref nRef, asyncResult);

                // Выдать четыре символа b в вызвавшей функции Main()
                int k= 0;
                while (k < 4)
                {
                        Console.Write ("c");
                        Thread.Sleep (10);
                        k++;
                }

                // Подождать окончания выполнения асинхронной функции  F()
                int n= del.EndInvoke (out  nOut, ref nRef, asyncResult);

                // Выдать значения результативных переменных на консоль
                Console.WriteLine ("\n n= " + n);
                Console.WriteLine (" nOut= " + nOut+ " nRef= " + nRef);
                Console.WriteLine (" asyncResult.IsCompleted= "
                                                + asyncResult.IsCompleted);
                Console.WriteLine (" asyncResult.CompletedSynchronously= "
                                        + asyncResult.CompletedSynchronously);
        }
}
/*
Result (Функция EndInvoke вызвана после цикла while):
acbcbcbc
 n= 3
 nOut= 3  nRef= 3
 asyncResult.IsCompleted= True
 asyncResult.CompletedSynchronously= False

 Result (Функция EndInvoke вызвана перед циклом while):
abbbccc
 n= 3
 nOut= 3  nRef= 3
 asyncResult.IsCompleted= True
 asyncResult.CompletedSynchronously= False*/
```

Пример 1.2. Асинхронный вызов с использованием функции обратного вызова.

```
//////////////
// C#  File  csK_Async_11
using System;
using System.Threading;

// Асинхронный делегат
delegate int DelN (string s,out int nOut, ref int nRef);
class A
{
      // Асинхронная функция
      public int F (string s,out int nOut, ref int nRef)
      {
            // Выдать три значения s в асинхронной функции
            int n= 0;
            while (n < 3)
            {
                  Console.Write (s);
                  Thread.Sleep (10);
                  n++;
            }
            //
            Console.WriteLine ("\n F: AppDomain.GetCurrentThreadId ( )= "
                                    + AppDomain.GetCurrentThreadId ( ));
            nOut= n; nRef= n;
            return n;
      }
}

class Test
{
      // Основная функция Main(), вызвавшая асинхронную функцию F()
      // с функцией обратного вызова
      static void Main (string[] args)
      {
            // Выдать идентификатор потока вызвавшей функции  Main()
            Console.WriteLine ("\nMain: AppDomain.GetCurrentThreadId ( )= "
                                       + AppDomain.GetCurrentThreadId ( ));
            A sA= new A ( );
            // Создать объект делегата для асинхронной функции
            DelN delN= new DelN (sA.F);

            // Отметить начало выполнения вызвавшей функции
            Console.WriteLine ("a");
            int nOut= 0;
            int nRef= 0;

            // Инициализировать асинхронный вызов с включением объекта
            // делегата AsyncCallback с функцией  обратного вызова (CallbackF)
            // и  данных (ref str) для функции EndInvoke().
            IAsyncResult asyncResult= delN.BeginInvoke   ("b",
                  out  nOut,  ref nRef, new AsyncCallback (CallbackF), delN);

            // Выдать четыре символа b в вызвавшей функции Main()
```

```
            int k= 0;
            while (k < 4)
            {
                  Console.Write ("c");
                  Thread.Sleep (10);
                  k++;
            }

            Console.WriteLine ("Нажать Enter для завершения приложения");
            Console.ReadLine ( );
      }

      static void CallbackF (IAsyncResult asyncResult)
      {
            // Получить ссылку на объект делегата DelN
            DelN del = (DelN) asyncResult.AsyncState;

            // Вызвать функцию EndInvoke() для получения результата
            int nOut= 0;
            int nRef= 0;
            int n = del.EndInvoke (out  nOut, ref nRef, asyncResult);
            Console.WriteLine ("n= " + n);
            Console.WriteLine (" nOut= " + nOut+ " nRef= " + nRef);
            Console.WriteLine (" asyncResult.IsCompleted= "
                                                + asyncResult.IsCompleted);
            Console.WriteLine (" asyncResult.CompletedSynchronously= "
                                    + asyncResult.CompletedSynchronously);
            Console.WriteLine
                        ("\nCallbackF: AppDomain.GetCurrentThreadI ()= "
                                    + AppDomain.GetCurrentThreadId ( ));
      }
}
/*
Result:
Main: AppDomain.GetCurrentThreadId()= 3060
a
cbcbcbc
F: AppDomain.GetCurrentThreadId ()= 3084
n= 3
nOut= 3  nRef= 3
asyncResult.IsCompleted= True
asyncResult.CompletedSynchronously= False
CallbackF: AppDomain.GetCurrentThreadId ()= 3084
Нажать Enter для завершения приложения
*/
```

2. Асинхронные события

При не асинхронных событиях выполнение функции, генерирующей событие, *приостанавливается до тех пор*, пока обработчики всех подписчиков (объектов-приёмников) этого события не будут выполнены. Порядок выполнения обработчиков определяется порядком подписки соответствующих объектов делегатов к объекту события издателя.

Пример 2.1. Обычное событие.

```
///////////////
// C#  File  csK_AsyncEv_1
using System;
using System.Threading;

delegate void DelEv (string s);       // Делегат события

class Source                          // Класс источника события (издатель)
{
      public event DelEv ev;          // Ссылка на объект события
      // Сгенерировать событие
      public void GenEv (string s)
      {
            DateTime d1 = DateTime.Now; // Время начала функции GenEv()
            Console.WriteLine ("GenEv: начало:" + d1);
            if (ev != null)           // Если на событие подписались, то
                  ev (s);             // выполнить
            d1 = DateTime.Now;              // Время конца функции GenEv()
            Console.WriteLine ("GenEv: конец :"  + d1);
      }
}

// Класс приёмника события (подписчик)
class Receiver
{
      int num;                        // Номер объекта
      public Receiver (int Num)
      {
            num= Num;
      }
      // Обработчик
      public void HandlerEv (string s )
      {
            DateTime d1 = DateTime.Now; // Время начала обработки события
            Console.WriteLine ("HandlerEv: receiver="+num+" начало:"+ d1);
            Thread.Sleep (10000);  // Поспать
            Console.WriteLine (s);   // Вывести полученное сообщение
            d1 = DateTime.Now;              // Время конца обработки события
            Console.WriteLine ("HandlerEv: receiver="+num " конец:"+ d1);
      }
}

class TestEv
{
      static void Main (string[] args)
      {
            Source source= new Source ( );        // Создать источник события
            Receiver receiver1= new Receiver (1);// Создать первый приёмник
            // Подписаться на событие
            source.ev += new DelEv (receiver1.HandlerEv);
            Receiver receiver2= new Receiver (2); // Создать второй приёмник
            // Подписаться на событие
            source.ev += new DelEv (receiver2.HandlerEv);
            source.GenEv ("Привет");              // Сгенерировать событие
```

```
            Console.WriteLine ("Нажать Enter для завершения приложения");
            Console.ReadLine ( );
        }
}
/*
Result:
GenEv: начало:                    22.07.2007 16:55:20
HandlerEv: receiver= 1 начало: 22.07.2007 16:55:20
Привет
HandlerEv: receiver= 1 конец:   22.07.2007 16:55:30
HandlerEv: receiver= 2 начало: 22.07.2007 16:55:30
Привет
HandlerEv: receiver= 2 конец:   22.07.2007 16:55:40
GenEv: конец:                     22.07.2007 16:55:40
Нажать Enter для завершения приложения
*/
```

Асинхронное событие решает проблему блокировки издателя: после генерирования события издатель продолжает функционировать в своём потоке, а *параллельно с ним* будут выполняться обработчики подписчиков, каждый в своём потоке.

Событие запускается с помощью функции BeginInvoke(), вызывая асинхронное выполнение обработчиков всех подписчиков.

```
public void GenAsyncEvent (string s)
{
    . . .
    if (evAsync != null)
       evAsync.BeginInvoke (s, null, null);
    . . .
}
```

Невозможно сразу инициировать выполнение всех обработчиков, поскольку функция BeginInvoke() инициирует выполнение *только одного* асинхронного вызова.

Выход – перебрать все объекты подписавшихся на событие делегатов и выполнить функцию BeginInvoke() применительно к каждому из них.

Извлечь список всех делегатов обработчиков, подписавшихся на событие, позволяет функция GetInvocationList() события. Известно, что событие – это особый делегат, принадлежащий объекту (издателю) и наследующий класс MulticastDelegate. Класс MulticastDelegate среди множества функций содержит нужную нам функцию GetInvocationList(), возвращающую список объектов делегатов, включённых в объект события в порядке их подсоединения.

```
public  sealed     override Delegate[] GetInvocationList ( );
```

Функция GetInvocationList() возвращает массив объектов делегатов типа Delegate. Поэтому при просмотре массива объектов делегатов требуется использовать преобразование типа.

```
public event DelAsyncEv evAsync;
public void GenAsyncEv  (string s)
{
   . . .
    if (evAsync != null)
    {
        Delegate [] delList= evAsync.GetInovationList  ( );
        foreach  (Delegate del in delList)
        {
            DelAsyncEv deleg=  (DelAsyncEv) del;
            deleg.BeginInvoke  (s, null, null);
        }
    }
   . . .
}
```

Пример 2.2. Асинхронное событие.

```
///////////////
// C#  File  csK_AsyncEv_3
using System;
using System.Threading;

delegate void DelAsyncEv (string s);       // Делегат события

class Source                               // Класс источника события (издатель)
{
       public event DelAsyncEv evAsync;  // Ссылка на объект события

       // Сгенерировать событие
       public void GenEv (string s)
       {
              DateTime d1 = DateTime.Now;
              Console.WriteLine ("GenEv: начало:" + d1);
              if (evAsync != null)           // Если на событие подписались, то
              {
                     // получить массив объектов делегатов, подписавшихся
                     // на событие,
                     Delegate [] delList= evAsync.GetInvocationList ( );
                     // просмотреть список подписавшихся объектов делегатов
                     // и выполнить их асинхронно
                     foreach (Delegate del in delList)
                     {
                        DelAsyncEv deleg= (DelAsyncEv) del;  // Текущий делегат
                        deleg.BeginInvoke (s, null, null);        // Выполнить
                     }
              }
              d1 = DateTime.Now;
              Console.WriteLine ("GenEv: конец:"   + d1);
       }
}

// Класс приёмника события (подписчик)
class Receiver
{
       int num;                                  // Номер объекта
```

```
        public Receiver (int Num)
        {
                num= Num;
        }

        // Обработчик
        public void HandlerEv (string s )
        {
                DateTime d1 = DateTime.Now;
                Console.WriteLine ("HandlerEv: receiver=”+num +" начало:"+ d1);
                Thread.Sleep (10000);           // Поспать
                Console.WriteLine (s);          // Вывести полученное сообщение
                d1 = DateTime.Now;
                Console.WriteLine ("HandlerEv: receiver="+num+" конец:"+ d1);
        }
}

class TestEv
{
        static void Main (string[] args)
        {
           Source source= new Source ( );          // Создать источник события

           Receiver receiver1= new Receiver (1);   // Создать первый приёмник
           // Подписаться на событие
           source.evAsync += new DelAsyncEv (receiver1.HandlerEv);

           Receiver receiver2= new Receiver (2);   // Создать второй приёмник
           // Подписаться на событие
           source.evAsync += new DelAsyncEv (receiver2.HandlerEv);

           source.GenEv ("Привет");                // Сгенерировать событие
           Console.WriteLine ("Нажать Enter для завершения приложения");
           Console.ReadLine ( );
        }
}
/*
Result:
GenEv: начало:                  22.07.2007 16:10:37
GenEv: конец:                   22.07.2007 16:10:37
Нажать Enter для завершения приложения
HandlerEv: receiver= 1 начало: 22.07.2007 16:10:37
HandlerEv: receiver= 2 начало: 22.07.2007 16:10:37
Привет
HandlerEv: receiver= 1 конец:   22.07.2007 16:10:47
Привет
HandlerEv: receiver= 2 конец:   22.07.2007 16:10:47
*/
```

ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ

1. Спроектировать и сконструировать программу согласно варианту индивидуального задания, выделив отдельные сущности в классы. Для взаимодействия между объектами классов использовать события.

2. Вызвать функции движения объектов асинхронно.

3. Вызвать функции движения объектов асинхронно, по их завершении вызывать функцию обратного вызова, которая выводит сообщение о том, что движение объекта завершено.

4. Вызвать движение объектов как реакцию на асинхронное событие.

Варианты индивидуальных заданий

1. Игра «Прятки». На форме присутствуют четыре круга. По нажатию на кнопку в окне появляется прямоугольник, который начинает двигаться в очерченный по его размерам центр окна. Как только прямоугольник достигнет центра, круги должны «убегать» каждый в свою сторону формы.

2. Игра «Футбол». На форме присутствуют три круга («защитники») и один прямоугольник («нападающий»). Прямоугольник при загрузке приложения начинает перемещаться от левой границы окна до правой и обратно. Как только «нападающий» переходит в «половину поля противника», «защитники» начинают перемещаться вправо, «на подмогу нападению». Как только прямоугольник начинает движение в обратную сторону, «защитники» начинают возвращаться на свои начальные позиции.

3. Игра «Эстафета». При загрузке приложения появляется красный круг, который движется слева направо, пока не достигнет границы окна – «финиша». После этого появляется синий круг, который получив «эстафетную палочку», движется в обратном направлении, вновь передавая на «финише» «эстафету» красному кругу.

4. Игра «Баскетбол». Вверху окна расположено прямоугольное табло, показывающее «0:0». По нажатию на правую/левую цифру табло слева появляется круг-«мяч», который движется к окружности-«корзине», расположенной справа. Как только «мяч» попадает в «корзину», значение в соответствующем элементе табло увеличивается.

5. Игра «Светофор». Посередине формы находится круг-«светофор» красного цвета, а с каждой стороны прямоугольники-«машины». Каждые 10 с цвет круга меняется с красного на зеленый или с зеленого на красный. Если «светофор» зеленый, «машины» движутся, иначе – останавливаются.

6. Игра «Водонос». После загрузки на форме через каждые 3 с начинают появляться голубые прямоугольники друг над другом. Как только их количество достигнет пяти, справа до прямоугольников «добегает» красный круг и снова «убегает» вправо, после чего прямоугольники исчезают, и «сосуд» начинает «наполняться» вновь.

7. Игра «Веселые старты». Слева через каждые 3 с появляются круги один над другим. Как только их становится пять, круги с разной скоростью начинают двигаться к правой границе. Как только один из кругов (самый быстрый) достигнет границы, все круги мгновенно замирают.

8. Игра «Пираты». Слева появляются прямоугольники и с разной скоростью начинают двигаться к середине формы. Как только все «шлюпки» достигнут середины экрана, от каждого прямоугольника отделяются по два небольших круга и с разной скоростью двигаются вправо. Как только «экипаж» достигнет правого края окна, «шлюпки» возвращаются на «корабль» (влево).

9. Игра «Урожай». Сверху начинают движение с разной скоростью красные круги-«яблоки» и останавливаются внизу форму. Как только их становится 10, справа движется прямоугольник-«садовник», который пробегает справа налево, после чего круги исчезают.

10. Игра «Дружная компания». Слева начинают появляться круги, выстраиваясь один за другим. Как только их наберется пять, круги начинают двигаться к правой границе экрана. По достижении ими правого края «компания» слева начинает «собираться» вновь.

11. Игра «Счеты». Справа начинают появляться желтые эллипсы с интервалом в 3 с. Как только их количество достигнет десяти, ниже появляется черный эллипс, а белые эллипсы исчезают.

12. Игра «Смена караула». Справа на форме 2 неподвижных синих круга. Слева появляются два красных круга и один прямоугольник перед ними – «караул» - которые начинают двигаться вправо. По достижении правой границы красные круги останавливаются, а синие круги и красный прямоугольник начинают движение справа налево.

13. Игра «Такси». Два круга начинают движение слева к центру формы. Достигнув центра, они останавливаются. Справа к ним начинает двигаться желтый прямоугольник-«такси». По достижении центра формы прямоугольником круги и прямоугольник начинают двигаться вниз формы.

14. Игра «Костер». Справа налево и обратно постоянно движется круг. По достижении центра в центре появляется коричневый прямоугольник. Как только таких прямоугольников станет 3, круг перестает двигаться, а над прямоугольниками появляется круг и начинает менять цвет с желтого на красный и обратно.

15. Игра «Лифт». Слева в центр движется черный круг. По достижению им центра он останавливается, а сверху вниз начинает двигаться желтый прямоугольник. По достижении им уровня круга прямоугольник и круг начинают свое движение вверх.

16. Игра «Магазин». В центре формы – синий прямоугольник. Снизу к нему приближается красный круг. Поравнявшись с прямоугольником, круг начинает менять цвет с красного на синий (10 раз). После этого прямоугольник делает то же самое. По окончании «диалога» круг «уходит» вверх.

17. Игра «Бомбардировщик». В центре формы находится синий прямоугольник. Слева движется серый эллипс. Достигнув центра формы, от эллипса вниз начинает двигаться черный круг. Поравнявшись с синим прямоугольником, он вызывает смену цвета прямоугольника на красный.

18. Игра «Охотник». Снизу формы находится зеленый круг. Слева начинает двигаться желтый прямоугольник. Как только он достигает центра, от зеленого прямоугольника начинает двигаться черный круг. Достигнув желтого прямоугольника он меняет его цвет на красный.

19. Игра «Сражение». Слева появляются 5 синих кругов с интервалом в 3 с. После появления последнего круга слева начинает двигаться синий прямоугольник и располагается перед кругами. После остановки прямоугольник окрашивается в красный цвет и вместе с кругами начинает движение вправо.