Делегаты. События

События

- **Понятие события.**
- **Синтаксис объявления события.**
- **Необходимость и особенности применения событий.**
- **Применение события для многоадресатного делегата.**
- **Использование событийных средств доступа.**

Событие (event) - это автоматическое уведомление о выполнении некоторого действия.

События работают следующим образом: объект, которому необходима информация о некотором событии, регистрирует обработчик для этого события. Когда ожидаемое событие происходит, вызываются все зарегистрированные обработчики.

Обработчики событий представляются делегатами.

Форма объявления события:

```
event событийный_делегат объект;
```

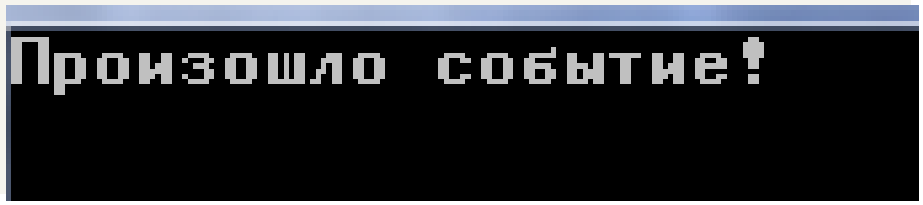
событийный_делегат - имя делегата, используемого для поддержки объявляемого события,

объект - это имя создаваемого событийного объекта.

Пример использования событий

```
// Объявляем делегат для события
    delegate void MyDel4Event();
    class Events
    {   // Метод, который выполняется при наступлении события
        static void handler()
        { Console.WriteLine("Произошло событие!");    }

       public static void Main()
        {   Class4Event someClass = new Class4Event();
            // Добавляем метод handler() в список события
            someClass.SomeEvent += new MyDel4Event(handler);
            // Генерируем событие (вызов события)
            someClass.OnSomeEvent();
            Console.ReadLine();
        }
    }
    // Объявляем класс события
    class Class4Event
    {   public event MyDel4Event SomeEvent;
        // Этот метод вызывается для генерирования события
        public void OnSomeEvent()
        { if (SomeEvent != null) SomeEvent(); } }
```



1. **Объявить делегат**, задающий сигнатуру метода, выполняемого при наступлении события.
2. **Объявить событие** типа делегата, заданного в п.1.
3. **Создать метод**, генерирующий событие.
4. **Зарегистрировать (подписать)** событие на метод, который соответствует сигнатуре делегата.
5. При определенных условиях **выполнить событие**.

Пример использования событий

```
public class Worker
    {   // объявление события
        public event delWork WorkEnded;

        public String Name;
        public String Post;
        public Worker(String Name, String Post)
        {   this.Name = Name;  this.Post = Post;   }

        // метод генерирующий событие
        protected virtual void OnEndWork()
        {  if (WorkEnded != null)  // проверка на подписку
               WorkEnded(Name);    // вызов события
        }

        public void Work()
        {  for (int i = 1; i <=8; i++)
            {
               if (i == 8)
                 OnEndWork(); // вызов метода, генер. событие
               else
                Console.WriteLine("Работаю {0} час!", i);
}}}
```

Пример использования событий (продолжение)

```
// объявление событийного делегата
public delegate void delWork(string message);

class Program
{  static void Main(string[] args)
   {
      Worker worker = new Worker("Alex","programmer");
      // подписка на обработчик события
      worker.WorkEnded += new delWork(Manager.GoHome);
      worker.Work();
   }
}

class Manager
{
   // метод, выступающий обработчиком события
   public static void GoHome(string str)
   {
    Console.WriteLine
           ("Молодцы, хорошо поработали! Получи премию, {0}!", str);
   }
}
```

Использование встроенного делегата EventHandler

Для упрощения процесса создания кода используется встроенный тип делегата, именуемый **EventHandler**.

Он используется для объявления обработчиков событий, которым не требуется дополнительная информация.

```
public delegate void EventHandler(object sender, EventArgs e);
```

Рекомендации по обработке событий в среде .NET Framework

1. C# позволяет программисту создавать события любого типа.
2. В целях компонентной совместимости со средой .NET Framework необходимо следовать рекомендациям Microsoft.

.NET-совместимые обработчики событий должны иметь следующую общую форму записи:

```
void handler(object   source,  EventArgs arg)
{
  // тело обработчика события
}
```

source - ссылка на объект, который генерирует событие.

arg - имеет тип EventArgs и содержит остальную информацию, необходимую обработчику.

```
public class Worker
{  // объявление события на основе делегата EventHandler
   public event EventHandler WorkEnded;

   public String Name;
   public String Post;

   public Worker(String Name, String Post)
   {   this.Name = Name; this.Post = Post;     }
   // метод генерирующий событие
   protected virtual void OnEndWork()
   {
      if (WorkEnded != null)
       // вызов события и передача ссылки на объект, который вызвал
       // событие и пустого обекта класса EventArgs.Empty
       {  WorkEnded(this, EventArgs.Empty); }
   }

   public void Work()
   {  for (int i = 1; i <= 8; i++)
       {   if (i == 8)
                OnEndWork(); // вызов метода
            else Console.WriteLine("Работаю {0} час!", i);
        }
}}
```

```
// событийного делегата не нужно т.к. используем EventHandler
class Program
{
  static void Main(string[] args)
    {
      Worker worker = new Worker("Alex", "programmer");
      Manager manager = new Manager();
      // подписка на обработчик события
      worker.WorkEnded += new EventHandler(manager.GoHome);
      worker.Work();
    }
}

class Manager
{
   // метод сообвествует сигнатуре делегата EventHandler
   public void GoHome(Object sender, EventArgs e)
    {  Worker worker = (Worker)sender;      // приведение типов !
       Console.WriteLine
         ("Рабочий день закончился! Иди домой, {0}!", worker.Name);
    }
}
```

Класс **EventArgs** не содержит полей, которые используются при передаче дополнительных данных обработчику; **он используется в качестве базового класса, из которого можно выводить класс, содержащий необходимые поля**.

Пример использования класса EventArgs

```
// Создаем класс, производный от класса EventArgs
    class MyEventArgs : EventArgs
    {
        public int eventnum;    // дополнительное поле
    }                           // для номера события
```

Для многих событий параметр типа **EventArgs** не используется.

```
public class WorkerEventArgs:EventArgs  // создание класс
{    public string Message { get; set; }
}
```

```
public class Worker
{  public event EventHandler WorkEnded;
   public String Name;   public String Post;
   public Worker(String Name, String Post)
    {   this.Name = Name;   this.Post = Post;     }

    protected virtual void OnEndWork()
    {   if (WorkEnded != null)
       { WorkerEventArgs workerArg = new WorkerEventArgs();
         workerArg.Message = "Можно еще поработать?!";
         // передача дополнительных параметров через WorkerEventArgs
         WorkEnded(this, workerArg);
     }}

    public void Work()
    {   for (int i = 1; i <= 8; i++)
        {  if (i == 8) OnEndWork();
             else Console.WriteLine("Работаю {0} час!", i);
        }
}}
```

```
class Program
{
  static void Main(string[] args)
   {
    Console.WriteLine("EventHandler");
    Worker worker = new Worker("Alex", "programmer");
    Manager manager = new Manager();
    worker.WorkEnded += new EventHandler(manager.GoHome);
    worker.Work();
} }

class Manager
{
// использование WorkerEventArgs для передачи дополнительных
// параметров через свойство Message
  public void GoHome(Object sender, EventArgs e)
  {
     Worker worker = (Worker)sender;
     WorkerEventArgs ww = (WorkerEventArgs)e;

    Console.WriteLine("Рабочий день закончился! Иди домой, {0}! {1}",
                                                 worker.Name, ww.Message);
  }
}
```

Пример события Windows Forms

```
private System.Windows.Forms.Button button1; // объект кнопки
```

```
// подписка на события Click и Move

button1.Click += new System.EventHandler(this.button1_Click);
button1.Move += new System.EventHandler(this.button1_Move);
```

```
// обработчик события на Click
private void button1_Click(object sender, EventArgs e)
 {
    textBox1.Text = "События!";
 }

// обработчик события на Click
private void button1_Move(object sender, EventArgs e)
 {

 }
```

Формы записей событий

Сокращенная форма записи событий

```
public event EventHandler WorkEnded;
```

Длинная форма записи событий

```
private event EventHandler workEnded;
public event EventHandler WorkEnded
{
    add
    {
        workEnded += value;
    }
    remove
    {
        workEnded -= value;
    }
}
```

- ключевые слова **add** и **remove** с событиями служат для добавления и удаления обработчика к делегату.

```
public event EventHandler<NewMailEventArgs> NewMail;

// 1. ЗАКРЫТОЕ поле делегата. инициализированное значением null
private EventHandler<NewMailEventArgs> NewMail = null;

// 2. ОТКРЫТЫЙ метод add_Xxx (где Xxx - это имя события )
// Позволяет объектам регистрироваться для получения уведомлений о событии
public void add_NewMail(EventHandler<NewMailEventArgs> value)
{   // Цикл и вызов CompareExchange - способ добавления
    // делегата в событие безопасным в отношении потоков путем
EventHandler<NewMailEventArgs>prevHandler ;
EventHandler<NewMailEventArgs>  newMail  =  this.NewMail;
do
{   prevHandler = newMail;
    EventHandler<NewMailEventArgs> newHandler =
                                (EventHandler<NewMailEventArgs>)Delegate.Combine(prevHandler.value);
    newMail = Interlocked.CompareExchange<EventHandler<NewMailEventArgs>>
                                                        (ref  this.NewMail, newHandler, prevHandler);
} while (newMail != prevHandler);
}

// 3. ОТКРЫТЫЙ метод remove_Xxx (где Xxx - это имя события )
// Позволяет объектам отменять регистрацию в качестве получателей уведомлений о событии
public void remove_NewMail(EventHandler<NewMailEventArgs> value)
{   // Цикл и вызов CompareExchange - способ удаления
    // делегата из события безопасным в отношении потоков путем
EventHandler<NewMailEventArgs>prevHandler ;
EventHandler<NewMailEventArgs>  newMail  =  this.NewMail;
do
{   prevHandler = newMail;
    EventHandler<NewMailEventArgs> newHandler =
                                    (EventHandler<NewMailEventArgs>)Delegate.Remove(prevHandler.value);
    newMail = Interlocked.CompareExchange<EventHandler<NewMailEventArgs>>
                                                          (ref  this.NewMail, newHandler, prevHandler);
} while (newMail != prevHandler);
}
```

Сравнивает два 32-разрядных целых числа со знаком на равенство и, если они равны, заменяет первое.

Слабые события

С помощью событий **издатель** (класс в котором реализовано событие) и **прослушиватель** (класс, содержащий обработчик события) соединяются напрямую. По этой причине может возникнуть проблема при сборке мусора. Например, если на прослушиватель никто больше не ссылается, все-таки остается ссылка со стороны издателя. Сборщик мусора не может очистить память, занимаемую прослушивателем, поскольку издатель удерживает ссылку и генерирует события для прослушивателя.

Это жесткое соединение может быть разрешено с использованием шаблона слабых событий и применения класса **WeakEventManager** в качестве посредника между издателем и прослушивателями.

```
// издатель
class Worker
{
    public event EventHandler EndWork;
    protected virtual void OnEndWorker()
    {
        if (EndWork != null)
            EndWork(this, EventArgs.Empty);
    }
    public void Work()
    {
        if (DateTime.Now.DayOfWeek == DayOfWeek.Monday)
            OnEndWorker();
    }
}

// прослушиватель
class TeamLeader
{
    public void GoToWeekEnd(object sender, EventArgs e)
    {
        Console.WriteLine("Go to WeekEnd");
    }
}
```

Использование WeakEventManager для реализации слабых событий

```
// Обобщенный диспетчер слабых событий
// для использования класс WeakEventManager подключаем WindowsBase.dll
// версия .NEt 4.5
static void Main(string[] args)
{
    Worker worker = new Worker();
    TeamLeader leader = new TeamLeader();
    //worker.EndWork += leader.GoToWeekEnd;
    WeakEventManager<Worker, EventArgs>
                .AddHandler(worker, "EndWork", leader.GoToWeekEnd);
    worker.Work();
    //worker.EndWork -= leader.GoToWeekEnd;
    WeakEventManager<Worker, EventArgs>
                .RemoveHandler(worker, "EndWork", leader.GoToWeekEnd);
    worker.Work();
    Console.Read();
}
```