
СОБЫТИЙНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ




ЦЕЛЬ

- Освоить событийно-управляемую парадигму программирования.

ПОНЯТИЕ

- Событийно-ориентированное программирование (англ. event-driven programming; в дальнейшем СОП) — парадигма программирования, в которой выполнение программы определяется событиями — действиями пользователя (клавиатура, мышь, сенсорный экран), сообщениями других программ и потоков, событиями операционной системы (например, поступлением сетевого пакета).





СОП можно также определить как способ построения компьютерной программы, при котором в коде (как правило, в головной функции программы) явным образом выделяется главный цикл приложения, тело которого состоит из двух частей: выборки события и обработки события.

Как правило, в реальных задачах оказывается недопустимым длительное выполнение обработчика события, поскольку при этом программа не может реагировать на другие события. В связи с этим при написании событийно-ориентированных программ часто применяют автоматное программирование.

СОБЫТИЙНО-ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ, КАК ПРАВИЛО, ПРИМЕНЯЕТСЯ В ТРЁХ СЛУЧАЯХ:

1. при построении пользовательских интерфейсов (в том числе графических);
2. при создании серверных приложений в случае, если по тем или иным причинам нежелательно порождение обслуживающих процессов;
3. при программировании игр, в которых осуществляется управление множеством объектов.

ПРИМЕНЕНИЕ СОБЫТИЙНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ В НАСТОЛЬНЫХ ПРИЛОЖЕНИЯХ


В современных языках программирования события и обработчики событий являются центральным звеном реализации графического интерфейса пользователя. Рассмотрим, к примеру, взаимодействие программы с событиями от мыши. Нажатие правой клавиши мыши вызывает системное прерывание, запускающее определенную процедуру внутри операционной системы. В этой процедуре происходит поиск окна, находящегося под курсором мыши. Если окно найдено, то данное событие посылается в очередь обработки сообщений этого окна. Далее, в зависимости от типа окна, могут генерироваться дополнительные события. Например, если окно является кнопкой (в Windows все графические элементы являются окнами), то дополнительно генерируется событие нажатия на кнопку. Отличие последнего события в том, что оно более абстрактно, а именно, не содержит координат курсора, а говорит просто о том, что было произведено нажатие на данную кнопку.

ОБРАБОТЧИК СОБЫТИЯ МОЖЕТ ВЫГЛЯДЕТЬ СЛЕДУЮЩИМ ОБРАЗОМ (НА ПРИМЕРЕ C#):

- `private void button1_Click(object sender, EventArgs e)`
- `{`
- `MessageBox.Show("Была нажата кнопка");`
- `}`
- Здесь обработчик события представляет собой процедуру, в которую передается параметр `sender`, как правило содержащий указатель на источник события. Это позволяет использовать одну и ту же процедуру для обработки событий от нескольких кнопок, различая их по этому параметру

ЯЗЫКИ СОБЫТИЙНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ

- В языке C# события реализованы как элемент языка и являются членами классов.
- Механизм событий здесь реализует шаблон проектирования Publisher/Subscriber. Пример
- объявления события:
- `public class MyClass`
- `{`
- `public event EventHandler MyEvent;`
- `}`
- Здесь `EventHandler` — делегат, определяющий тип процедуры обработчика событий. Подписка на событие производится следующим образом: `myClass.MyEvent += new EventHandler(Handler)`; Здесь `myClass` — экземпляр класса `MyClass`, `Handler` — процедура-обработчик. Событие может иметь неограниченное количество обработчиков. При добавлении обработчика события он добавляется в специальный стек, а при возникновении события вызываются все обработчики по их порядку в стеке. Отписка от события, то есть удаление обработчика производится аналогично, но с использованием оператора «-=».



Разные языки программирования поддерживают СОП в разной степени. Наиболее полной поддержкой событий обладают следующие языки (неполный список):

- о Perl (события и демоны DAEMON, и их приоритеты PRIO),
- о PHP
- о Java,
- о Delphi (язык программирования),
- о ActionScript 3.0,
- о C# (события event),
- о JavaScript (действия пользователя).

Остальные языки, в большей их части, поддерживают события как обработку исключительных ситуаций.

ПРИМЕР УПРАВЛЕНИЯ СОБЫТИЯМИ ПОСРЕДСТВОМ ДЕЛЕГАТА: //

ОБЪЯВЛЕНИЕ

ДЕЛЕГАТА, НА ОСНОВЕ КОТОРОГО БУДЕТ // ОПРЕДЕЛЕНО СОБЫТИЕ.

```
Delegate void MyEventHandler () ;

// Объявление класса, в котором инициируется событие, class MyEvent
{ public event MyEventHandler activate;

// В этом методе инициируется событие.

Public void fire()
{ if (activate != null) activate(); }

}

Class X
{

Public void Xhandler()
{

Console.WriteLine(«Событие получено объектом класса X. «) ;

}

}

Class Y
{

Public void YhandlerO
{

Console.WriteLine(«Событие получено объектом класса Y.»);

}

}

Class EventDemo
{

Static void handler()
{
```

```
Console.WriteLine(«Событие получено объектом класса EventDemo.»)

}

Public static void Main()

{

MyEvent evt = new MyEvent();

X xOb = new X();

Y yOb = new Y ();

// Добавление методов handler (), Xhandler 0

// и YhandlerO в цепочку обработчиков события.

Evt.activate += new MyEventHandler(handler);

Evt.activate += new MyEventHandler(xOb.Xhandler);

Evt.activate += new MyEventHandler(yOb.Yhandler);

Evt.fire();

Console.WriteLine();

Evt.activate -= new MyEventHandler(xOb.Xhandler);

Evt.fire();

}

}
```

ИСКЛЮЧИТЕЛЬНЫЕ СИТУАЦИИ

- Исключительная ситуация (или, короче, исключение) представляет собой ту или иную ошибку или отклонение от обычного сценария, происходящее во время выполнения программы. С помощью подсистемы обработки исключений языка C# можно обрабатывать такие ошибки, не вызывая аварийного завершения программы.
- Для обработки исключений в языке C# применяются следующие ключевые слова: try, catch, throw, finally. Перечисленные ключевые слова образуют взаимосвязанную подсистему, в которой использование одного из ключевых слов влечет за собой использование других. Обработка исключений в языке C# реализуется на основе операторных блоков try и catch.

ОПИСАНИЕ БЛОКА TRY И CATCH

- Синтаксис:
- Try {
- Блок_кода_для_которого_выполняется_мониторинг_ошибок
- Catch (Exception exObj) {
- Обработчик_исключений_Exception1 }
- Catch (Exception2 exObj) {
- Обработчик_исключений_Exception2 }

ОСНОВНЫЕ СИСТЕМНЫЕ ИСКЛЮЧЕНИЯ ПРИВЕДЕНЫ В СЛЕДУЮЩЕЙ ТАБЛИЦЕ:

- `ArrayTypeMismatchException` - Тип сохраненного значения несовместим с типом массива
- `DivideByZeroException` - Предпринята попытка деления на ноль
- `IndexOutOfRangeException` - Индекс массива выходит за пределы диапазона
- `InvalidCastException` - Некорректное преобразование в процессе выполнения
- `OutOfMemoryException` - Вызов `new` был неудачным из-за недостатка памяти
- `Overflow/Exception` - Переполнение при выполнении арифметической операции
- `StackOverflowException` - Переполнение стека

ВЫВОДЫ

Событийно-ориентированное программирование, как правило, применяется в следующих случаях:

- построение пользовательских интерфейсов (в том числе графических);
- создание серверных приложений;
- моделирование сложных систем;
- параллельные вычисления;
- автоматические системы управления, SCADA;
- программирование игр, в которых осуществляется управление множеством объектов.

Примеры реализаций событийно-ориентированного программирования

- Веб-серверы: Node.js, nginx, lighttpd
- Прокси-серверы: Squid