Семинар 2. Делегаты

Чуйкин Николай Константинович

22 января 2020 г.

Делегаты

- Делегаты
- Передача параметров в делегат и возврат значения
- Подходы к решению задач
- Callback
- `StreamWriter`

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:
  - Одноадресные *в C# не поддерживается*

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:
  - Одноадресные *в C# не поддерживается*
  - Многоадресные

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:
  - Одноадресные *в C# не поддерживается*
  - Многоадресные
- По типу возвращаемого значения:

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:
  - Одноадресные *в C# не поддерживается*
  - Многоадресные
- По типу возвращаемого значения:
  - Возвращающие `void`

Делегаты

Делегаты - встроенный механизм языка C# для передачи ссылки на метод
Не следует путать делегат-тип и делегат-экземпляр
Делегаты можно подразделить:

- По количеству методов на которые они ссылаются:
  - Одноадресные *в C# не поддерживается*
  - Многоадресные
- По типу возвращаемого значения:
  - Возвращающие `void`
  - Возвращающие значение отличное от `void`

Делегат-тип

Делегат тип описывает на какие именно методы может ссылаться конкретный делегат.
Для обявления делегата типа используется ключевое слово `delegate`
Примеры объявления делегатов :

- `delegate void MyDel();`
- `delegate double Function2(double x);`
- `delegate double Function3(double x, double y);`
- `delegate void MyRefDelegate(ref int x);`
- `delegate int MyCompareDelegate(int x,int y);`

Делегат-тип

На самом деле основные типы делегатов уже определены, поэтому лучше использовать стандартные а не создавать свои
Примеры стандартных делегатов вместо собственных :

- `Action`
- `Func<double,double>`
- `Func<double,double,double>`
- С `ref` тип надо создавать
- `Comparison<int>`

Делегат-экземпляр

Делегат-экземпляр это ссылка на конкретную функцию.
Для создание используется обычный синтакс объявления и создания объектов:

```
Function2 instanse = new Function2(Math.Sin);
```

Делегат-экземпляр

Делегат-экземпляр это ссылка на конкретную функцию.
Для создание используется обычный синтакс объявления и создания объектов:

```
Function2 instanse = new Function2(Math.Sin);
```

Но этот синтаксис может быть упрощен до:

```
Function2 instanse = Math.Sin;
```

Делегат-экземпляр

Делегат-экземпляр это ссылка на конкретную функцию.
Для создание используется обычный синтакс объявления и создания объектов:

```
Function2 instanse = new Function2(Math.Sin);
```

Но этот синтаксис может быть упрощен до:

```
Function2 instanse = Math.Sin;
```

Для вызова функции необходимо написать:

```
double x = instanse.Invoke(Math.PI);
```

или

```
double x = instanse(Math.PI);
```

Делегат-экземпляр

У экземпляра делегата есть следующие поля, методы и допустимые операции:

- `=`
- `+=`
- `-=`
- `Invoke()`

Делегат-экземпляр

У экземпляра делегата есть следующие поля, методы и допустимые операции:

- `=`
- `+=`
- `-=`
- `Invoke()`
- `GetInvocationList()`
- `DynamicInvoke()`
- `target`

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
Console.WriteLine(instanse(Math.PI));
```

Что будет выведено на экран?

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
Console.WriteLine(instanse(Math.PI));
```

Что будет выведено на экран?

-1

Так как возвращается значение только последнего метода

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
foreach(Delegate method in instanse.GetInvocationList())
    Console.WriteLine(method.Invoke(Math.PI));
```

Будет ли работать этот метод?

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
foreach(Delegate method in instanse.GetInvocationList())
    Console.WriteLine(method.Invoke(Math.PI));
```

Будет ли работать этот метод?

Нет, так как базовый класс делегат не знает сигнатуру метода

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
foreach(Delegate method in instanse.GetInvocationList())

    Console.WriteLine(method.DynamicInvoke(Math.PI));
```

Это работает, но проверка соответствия типов происходит на этапе...

Делегат-экземпляр

```
Function2 instanse = Math.Sin;
instanse += Math.Cos;
foreach(Delegate method in instanse.GetInvocationList())

    Console.WriteLine(method.DynamicInvoke(Math.PI));
```

Это работает, но проверка соответствия типов происходит на этапе...
Исполнения

Подходы к решению задач

Как сделать так, чтобы делегат возвращал сумму или композицию значений всех методов, при том, чтобы проверка типов производилась на этапе компиляции?

Подходы к решению задач

Как сделать так, чтобы делегат возвращал сумму или композицию значений всех методов, при том, чтобы проверка типов производилась на этапе компиляции?

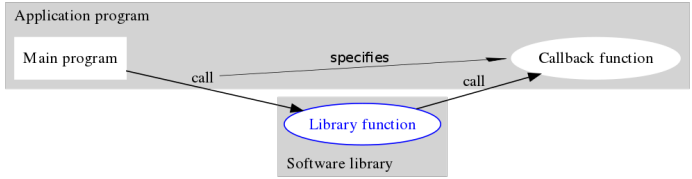
- Для композиции используется передача параметра по ссылке

Подходы к решению задач

Как сделать так, чтобы делегат возвращал сумму или композицию значений всех методов, при том, чтобы проверка типов производилась на этапе компиляции?

- Для композиции используется передача параметра по ссылке
- Для суммирования можно использовать массив экземпляров делегата

Callback


Application program
Main program
call
specifies
Callback function
call
Library function
Software library

StreamWriter

`StreamWriter` - тип позволяющий удобно работать с файлами, как с ”консолью”.

Для создания объекта необходимо указать в конструкторе название файла:

```
StreamWriter sw = new StreamWriter("output.txt")
```

Но тогда обязательно необходимо закрыть файл, иначе результат может не сохраниться:

```
sw.Close()
```

А что если между этими строками произойдёт исключение?

StreamWriter

`StreamWriter` - тип позволяющий удобно работать с файлами, как с ”консолью”.

Для создания объекта необходимо указать в конструкторе название файла:

```
StreamWriter sw = new StreamWriter("output.txt")
```

Но тогда обязательно необходимо закрыть файл, иначе результат может не сохраниться:

```
sw.Close()
```

А что если между этими строками произойдёт исключение?

Результат может не сохраниться

StreamWriter

Тогда код можно переписать так:

```
try {
     StreamWriter sw = new StreamWriter("output.txt")
     ...
}
finally {
     sw.Close()
}
```

Но такой код пришлось бы писать постоянно, поэтому разработчики предумали...

StreamWriter

Тогда код можно переписать так:

```
using( StreamWriter sw = new StreamWriter("output.txt")){
     ...
}
```

Спасибо за внимание