Ответы на 7 лабараторную с#

1. Расскажите как генерируется исключение.

Обычно система сама генерирует исключения при определенных ситуациях, например, при делении числа на ноль. Но язык C# также позволяет генерировать исключения вручную с помощью оператора throw. То есть с помощью этого оператора мы сами можем создать исключение и вызвать его в процессе выполнения.

Например, в нашей программе происходит ввод строки, и мы хотим, чтобы, если длина строки будет больше 6 символов, возникало исключение:

static void Main(string[] args)

{

    try

    {

        Console.Write("Введите строку: ");

        string message = Console.ReadLine();

        if (message.Length > 6)

        {

            throw new Exception("Длина строки больше 6 символов");

        }

    }

    catch (Exception e)

    {

        Console.WriteLine($"Ошибка: {e.Message}");

    }

    Console.Read();

}

После оператора throw указывается объект исключения, через конструктор которого мы можем передать сообщение об ошибке. Естественно вместо типа Exception мы можем использовать объект любого другого типа исключений.

Затем в блоке catch сгенерированное нами исключение будет обработано.

Подобным образом мы можем генерировать исключения в любом месте программы. Но существует также и другая форма использования оператора throw, когда после данного оператора не указывается объект исключения. В подобном виде оператор throw может использоваться только в блоке catch:

try

{

    try

    {

        Console.Write("Введите строку: ");

        string message = Console.ReadLine();

        if (message.Length > 6)

        {

            throw new Exception("Длина строки больше 6 символов");

        }

    }

    catch

    {

        Console.WriteLine("Возникло исключение");

        throw;

    }

}

catch (Exception ex)

{

    Console.WriteLine(ex.Message);

}

2. Расскажите методику обработки исключений.выше

3. Какое ключевое слово служит для обозначения блока кода, в котором

можно генерировать исключение?

Throw

4. Какие ключевые слова используются для обработки и генерации

исключений? Расскажите об механизме обработке исключения?

Иногда при выполнении программы возникают ошибки, которые трудно предусмотреть или предвидеть, а иногда и вовсе невозможно. Например, при передачи файла по сети может неожиданно оборваться сетевое подключение. такие ситуации называются исключениями. Язык C# предоставляет разработчикам возможности для обработки таких ситуаций. Для этого в C# предназначена конструкция try...catch...finally.

try

{

}

catch

{

}

finally

{

}

При использовании блока try...catch..finally вначале выполняются все инструкции в блоке try. Если в этом блоке не возникло исключений, то после его выполнения начинает выполняться блок finally. И затем конструкция try..catch..finally завершает свою работу.

Если же в блоке try вдруг возникает исключение, то обычный порядок выполнения останавливается, и среда CLR начинает искать блок catch, который может обработать данное исключение. Если нужный блок catch найден, то он выполняется, и после его завершения выполняется блок finally.

Если нужный блок catch не найден, то при возникновении исключения программа аварийно завершает свое выполнение.

Рассмотрим следующий пример:

class Program

{

    static void Main(string[] args)

    {

        int x = 5;

        int y = x / 0;

        Console.WriteLine($"Результат: {y}");

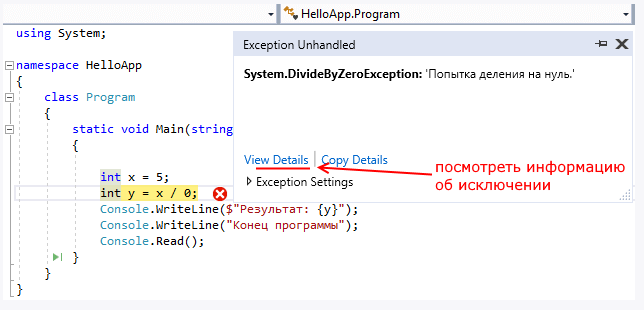
        Console.WriteLine("Конец программы");

        Console.Read();

    }

}

В данном случае происходит деление числа на 0, что приведет к генерации исключения. И при запуске приложения в режиме отладки мы увидим в Visual Studio окошко, которое информирует об исключении:



В этом окошке мы видим, что возникло исключение, которое представляет тип System.DivideByZeroException, то есть попытка деления на ноль. С помощью пункта View Details можно посмотреть более детальную информацию об исключении.

И в этом случае единственное, что нам остается, это завершить выполнение программы.

Чтобы избежать подобного аварийного завершения программы, следует использовать для обработки исключений конструкцию try...catch...finally. Так, перепишем пример следующим образом:

{

    static void Main(string[] args)

    {

        try

        {

            int x = 5;

            int y = x / 0;

            Console.WriteLine($"Результат: {y}");

        }

        catch

        {

            Console.WriteLine("Возникло исключение!");

        }

        finally

        {

            Console.WriteLine("Блок finally");

        }

        Console.WriteLine("Конец программы");

        Console.Read();

    }

}

В данном случае у нас опять же возникнет исключение в блоке try, так как мы пытаемся разделить на ноль. И дойдя до строки

int y = x / 0;

выполнение программы остановится. CLR найдет блок catch и передаст управление этому блоку.

После блока catch будет выполняться блок finally.

Таким образом, программа по-прежнему не будет выполнять деление на ноль и соответственно не будет выводить результат этого деления, но теперь она не будет аварийно завершаться, а исключение будет обрабатываться в блоке catch.

Следует отметить, что в этой конструкции обязателен блок try. При наличии блока catch мы можем опустить блок finally:

try

{

    int x = 5;

    int y = x / 0;

    Console.WriteLine($"Результат: {y}");

}

catch

{

    Console.WriteLine("Возникло исключение!");

}

И, наоборот, при наличии блока finally мы можем опустить блок catch и не обрабатывать исключение:

try

{

    int x = 5;

    int y = x / 0;

    Console.WriteLine($"Результат: {y}");

}

finally

{

    Console.WriteLine("Блок finally");

}

Однако, хотя с точки зрения синтаксиса C# такая конструкция вполне корректна, тем не менее, поскольку CLR не сможет найти нужный блок catch, то исключение не будет обработано, и программа аварийно завершится.

5. Что будет, если в программе нет предложения catch, способного

обработать исключение?

///

6. Что такое фильтры исключения? Приведите пример

ильтры исключений срабатывают, если при выполнении метода действия будет выброшено необработанное исключение.

С одной стороны, мы могли поместить всю логику выполнения метода в блок try...catch и отследить исключение. Однако область работы фильтров исключения несколько шире. Они позволяют отследить не только исключения, возникающие в самом методе, но исключения, генерируемые результатами действий, а также другими применяемыми к данному действию фильтрами. В этом и состоит мощь данного типа фильтров.

Все фильтры исключений должны применять интерфейс IExceptionFilter:

public interface IExceptionFilter

{

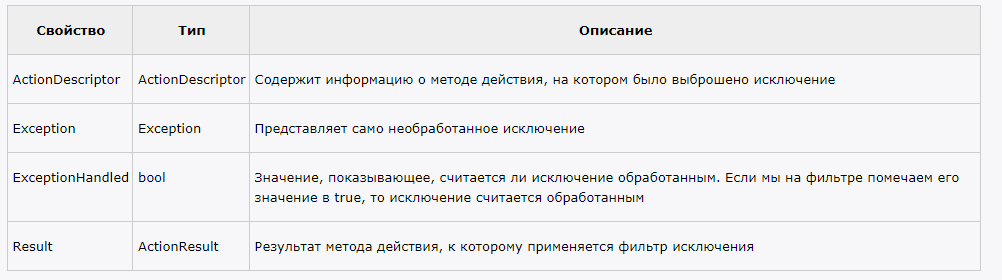
    void OnException(ExceptionContext filterContext);

}

И если вдруг приложение выбрасывает необрабатываемое исключение, то фильтр вызывает метод OnException.

Передаваемый в этот метод параметр - ExceptionContext является объектом, производным от ControllerContext. Поэтому из него можно извлечь как специфичную для фильтра информацию, так и общую информацию о запросе.

В частности класс ExceptionContext обладает следующими свойствами, которые позволяют получить информацию об исключении:



С помощью свойства Exception мы можем получить доступ к выбрасываемому исключению.

Установив свойство ExceptionHandled в true, фильтр тем самым помечает исключение как обработанное.

С помощью свойства Result фильтр управляет результатом действий. Общераспространенной практикой в данном случае является перенаправление пользователя на страницу ошибки или отображение ошибки на экране.

Теперь создадим простенький фильтр, который будет обрабатывать исключение IndexOutOfRangeException:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Mvc;

namespace Filters.Filters

{

    public class IndexException : FilterAttribute, IExceptionFilter

    {

        public void OnException(ExceptionContext exceptionContext)

        {

            if (!exceptionContext.ExceptionHandled && exceptionContext.Exception is IndexOutOfRangeException)

            {

                exceptionContext.Result = new RedirectResult("/Content/ExceptionFound.html");

                exceptionContext.ExceptionHandled = true;

            }

        }

    }

}

Здесь в методе OnException первым делом мы проверяем, не установлено ли значение свойства ExceptionHandled. Если оно установлено в true, следовательно, какой-то другой фильтр исключений уже обработал данное исключение. Также проверяется тип исключения. Поскольку мы ловим в данном случае только исключения типа IndexOutOfRangeException, следовательно, нас только они интересуют.

Далее мы устанавливаем результат метода, к которому применен фильтр. Предполагается, что в проекте в каталоге Content у нас находится некоторая страница ExceptionFound.html, которая отображает пользователю сообщение об ошибке.

В данном случае важно пометить исключение как обработанное: exceptionContext.ExceptionHandled = true. Иначе, если мы этого не сделаем, то мы можем увидеть в браузере все диагностическое сообщение об ошибке, которое обычно посылает фреймворк в ответ клиенту.

Пример использования:

[IndexException]

public ActionResult Index()

{

    int[] mas = new int[2];

    mas[6] = 4;

    return View();

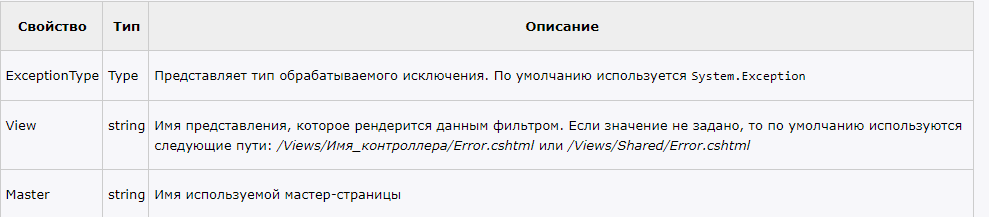
}

В данном случае метод Index выбросит необработанное исключение, и оно будет объектом типа IndexOutOfRangeException, а пользователь будет перенаправлен на страницу ExceptionFound.htm.

Подобным образом мы можем обработать и другие типы исключений.

HandleErrorAttribute. Встроенная обработка исключений.

Создавать свои фильтры исключений необязательно, так как во фреймворке имеется встроенная реализация интерфейса IExceptionFilter - атрибут HandleErrorAttribute. Он имеет ряд свойств, с помощью которых мы можем произвести гибкую настройку фильтра:



При обработке исключения фильтр исключений посылает статусный код HTTP 500 и генерирует указанное в свойстве View представление. Например, используем предыдущий пример с фильтром исключений, применив встроенную реализацию:

[HandleError(ExceptionType = typeof(System.IndexOutOfRangeException), View = "ExceptionFound")]

public ActionResult About()

{

    int[] mas = new int[2];

    mas[6] = 4;

    return View();

}

В данном случае очевидно, что на строке mas[6] = 4; будет выброшено исключение. В режиме отладки у вас приостановится выполнение программы, тогда вы можете нажать на кнопку Continue на панели инструментов. Здесь опять мы обрабатываем исключение типа IndexOutOfRangeException, и при возникновении такового генерируем в ответ представление ExceptionFound.cshtml. Данное представление должно находиться в проекте в каталоге Views/Имя\_контроллера/.

Сразу надо сказать, что если вы хотите при разработке видеть обрабатываемые фильтром HandleErrorAttribute, то надо включить данную функциональность в файле конфигурации web.config с помощью тега <customErrors mode="On" />:

<system.web>

    <customErrors mode="On"  />

    <!-- далее остальное содержание узла system.web -->

</system.web>

7. Могут ли исключения быть вложенными?

Да

8. Какой синтаксис нужно использовать в C# для отлова любого возможного

исключения?

Блок catch, отлавливающий исключения типа System.Exception. Также можно просто использовать catch{}, не указывая тип исключения.

9. Чем следует руководствоваться при размещении обработчиков

исключения?

Выше

Мб:

Руководствоваться нужно определением типа исключения. В зависимости от того, что вы хотите обрабатывать или видеть нужно и наследоваться от нужного класса.

10.Что будет выведено на консоль в результате выполнения фрагмента

листинга?

static void Main(string[] args)

{

string[] str = new string[5];

try {

str[4] = "anything";

Console.WriteLine("It's OK");

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine("IndexOutOfRangeException");

}

catch (Exception e)

{

Console.WriteLine("Exception");

}

}

}

It s OK

11.Как повторно сгенерировать то же самое исключение в блоке

обработчике catch?

///

На с++

Если возникает необходимость снова сгенерировать исключения из блока, который обрабатывает исключения, можно сделать это путем вызова throw без указания исключения. В результате теку­щее исключение будет передано во внешнюю последовательность try/catch обработки исключений. Причиной для этого может послужить желание обрабатывать исключения несколькими обработ­чиками. Например, один обработчик может заниматься одним аспектом исключения, а второй обработчик — другим. Исключение может быть снова сгенерировано или изнутри блока catch, или из функции, вызванной в этом блоке. Когда повторно генерируется исключение, оно не бу­дет перехвачено той же самой инструкцией catch. Оно будет распространяться до следующей внешней инструкции catch. Следующая программа иллюстрирует повторную генерацию исключе­ния. В ней повторно генерируется исключение типа char\*.

// пример повторной генерации исключения

#include <iostream.h>

void Xhandler()

{

try {

throw "hello"; // генерация char \*

}

catch (char \*) { // перехват char \*

cout << "Caught char \* inside Xhandler\n";

throw; // повторная генерация char \* извне функции

}

}

int main()

{

cout << "Start\n";

try{

Xhandler();

}

catch(char \*) {

cout << "Caught char \* inside main\n";

}

cout << "End";

return 0;

}

12..Какие методы содержаться в классе Exception? Где и как их можно

использовать?

Базовым для всех типов исключений является тип Exception. Этот тип определяет ряд свойств, с помощью которых можно получить информацию об исключении.

InnerException: хранит информацию об исключении, которое послужило причиной текущего исключения

Message: хранит сообщение об исключении, текст ошибки

Source: хранит имя объекта или сборки, которое вызвало исключение

StackTrace: возвращает строковое представление стека вызывов, которые привели к возникновению исключения

TargetSite: возвращает метод, в котором и было вызвано исключение

static void Main(string[] args)

{

    try

    {

        int x = 5;

        int y = x / 0;

        Console.WriteLine($"Результат: {y}");

    }

    catch (Exception ex)

    {

        Console.WriteLine($"Исключение: {ex.Message}");

        Console.WriteLine($"Метод: {ex.TargetSite}");

        Console.WriteLine($"Трассировка стека: {ex.StackTrace}");

    }

    Console.Read();

}