



Строки и исключения



Дмитрий Шалымов

Ведущий курса

dmitry.shalymov@veeamacademy.ru

СИМВОЛЫ

Символы в .NET всегда представлены 16-разрядными кодами

Юникод

MinValue = '\0'

MaxValue = '\uffff'

```
Double d; // '\u0033' – это "цифра 3"
d = Char.GetNumericValue('\u0033'); // Параметр '3'
// даст тот же результат
Console.WriteLine(d.ToString()); // Выводится "3"
// '\u00bc' – это "простая дробь одна четвертая ('1/4')"
d = Char.GetNumericValue('\u00bc');
Console.WriteLine(d.ToString()); // Выводится "0.25"
// 'A' – это "Латинская прописная буква A"
d = Char.GetNumericValue('A');
Console.WriteLine(d.ToString()); // Выводится "-1"
```



Тип System.String

Ссылочный тип

Реализует интерфейсы:

- IComparable/IComparable<String>
- ICloneable
- IConvertible
- IEnumerable/IEnumerable<Char>
- IEquatable<String>

```
String s = new String("Hi there."); // Ошибка
String s = "Hi there.";
```

```
// Конкатенация трех литеральных строк образует одну литеральную строку
String s = "Hi" + " " + "there.";
```



Неизменяемые строки

Созданную однажды строку нельзя сделать длиннее или короче, в ней нельзя изменить ни одного символа

```
if (s.ToUpperInvariant().Substring(10, 21).EndsWith("EXE"))  
{  
    ...  
}
```

Отпадает проблема синхронизации потоков при работе со строками

Класс String является запечатанным



Сравнение строк

```
Boolean Equals(String value, StringComparison comparisonType)
static Boolean Equals(String a, String b, StringComparison comparisonType)

static Int32 Compare(String strA, String strB,
StringComparison comparisonType)
static Int32 Compare(string strA, string strB,
Boolean ignoreCase, CultureInfo culture)
static Int32 Compare(String strA, String strB,
CultureInfo culture, CompareOptions options)
static Int32 Compare(String strA, Int32 indexA, String strB, Int32 indexB,
Int32 length, StringComparison comparisonType)
static Int32 Compare(String strA, Int32 indexA, String strB, Int32 indexB,
Int32 length, CultureInfo culture, CompareOptions options)
static Int32 Compare(String strA, Int32 indexA, String strB, Int32 indexB,
Int32 length, Boolean ignoreCase, CultureInfo culture)

Boolean StartsWith(String value, StringComparison comparisonType)
Boolean StartsWith(String value,
Boolean ignoreCase, CultureInfo culture)

Boolean EndsWith(String value, StringComparison comparisonType)
Boolean EndsWith(String value, Boolean ignoreCase, CultureInfo culture)
```

```
public enum StringComparison
{
    CurrentCulture = 0,
    CurrentCultureIgnoreCase = 1,
    InvariantCulture = 2,
    InvariantCultureIgnoreCase = 3,
    Ordinal = 4,
    OrdinalIgnoreCase = 5
}
```



Лингвистически корректные сравнения

System.Globalization.CultureInfo:

- en-US означает американскую (США) версию английского языка
- en-AU — австралийскую версию английского языка
- de-DE — германскую версию немецкого языка

CurrentUICulture

CurrentCulture



Региональные стандарты и сортировка

```
String s1 = "Strasse";
String s2 = "Straße";
Boolean eq;
// CompareOrdinal возвращает ненулевое значение
eq = String.Compare(s1, s2, StringComparison.Ordinal) == 0;
Console.WriteLine("Ordinal comparison: '{0}' {2} '{1}'", s1, s2,
    eq ? "==" : "!=");
// Сортировка строк для немецкого языка (de) в Германии (DE)
CultureInfo ci = new CultureInfo("de-DE");
// Compare возвращает нуль
eq = String.Compare(s1, s2, true, ci) == 0;
Console.WriteLine("Cultural comparison: '{0}' {2} '{1}'", s1, s2,
    eq ? "==" : "!=");
```

```
Ordinal comparison: 'Strasse' != 'Straße'
Cultural comparison: 'Strasse' == 'Straße'
```



Интернирование строк

При инициализации CLR создает внутреннюю хеш-таблицу, в которой ключами являются строки, а значениями — ссылки на строковые объекты в управляемой куче

```
public static String Intern(String str);  
public static String IsInterned(String str);
```

По умолчанию при загрузке сборки CLR интернирует все литеральные строки, описанные в метаданных сборки

```
String s1 = "Hello";  
String s2 = "Hello";  
Console.WriteLine(Object.ReferenceEquals(s1, s2)); // Должно быть 'False'  
  
s1 = String.Intern(s1);  
s2 = String.Intern(s2);  
Console.WriteLine(Object.ReferenceEquals(s1, s2)); // 'True'
```



Интернирование строк

```
private static Int32 NumTimesWordAppearsEquals(String word, String[] wordlist)
{
    Int32 count = 0;
    for (Int32 wordnum = 0; wordnum < wordlist.Length; wordnum++)
    {
        if (word.Equals(wordlist[wordnum], StringComparison.Ordinal))
            count++;
    }
    return count;
}
```

```
private static Int32 NumTimesWordAppearsIntern(String word, String[] wordlist)
{
    word = String.Intern(word);
    Int32 count = 0;
    for (Int32 wordnum = 0; wordnum < wordlist.Length; wordnum++)
    {
        if (Object.ReferenceEquals(word, wordlist[wordnum]))
            count++;
    }
    return count;
}
```



Прочие операции со строками

- Clone
- Copy
- CopyTo
- ToString

```
string s = "Hi from St-Petersburg!";
object clone = s.Clone();
// True True
Console.WriteLine(
    $"Equals:{s.Equals(clone)}, Ref Equals: {Object.ReferenceEquals(s, clone)}");

// True False
string copy = string.Copy(s);
Console.WriteLine(
    $"Equals:{s.Equals(copy)}, Ref Equals: {Object.ReferenceEquals(s, copy)}");

char[] destination = { '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*',
    '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*', '*' };

s.CopyTo(0, destination, 2, s.Length);
// "***Hi from St-Petersburg!*"
Console.WriteLine(destination);

// True True
string toString = s.ToString();
Console.WriteLine(
    $"Equals:{s.Equals(toString)}, Ref Equals: {Object.ReferenceEquals(s, toString)}");
```

Эффективное создание строк

StringBuilder - общедоступный конструктор для String

Внутренние поля:

- Максимальная емкость
- Емкость
- Массив СИМВОЛОВ

```
StringBuilder sb = new StringBuilder();  
String s = sb.AppendFormat("{0} {1}", "Hi There!", "from Saint-Petersburg").  
    Replace(' ', '-').Remove(4, 3).ToString();
```



Получение строкового представления объекта

Реализация ToString в типе System.Object просто возвращает полное имя типа объекта

```
public interface IFormattable
{
    String ToString(String format, IFormatProvider formatProvider);
}
```



Региональные стандарты

Региональные стандарты влияют на форматирование чисел (включая денежные суммы, целые числа, числа с плавающей точкой и проценты), дат и времени

```
public interface IFormatProvider
{
    Object GetFormat(Type formatType);
}
```

```
Decimal price = 123.54M;
String s = price.ToString("C", new CultureInfo("vi-VN"));
```



Форматирование нескольких объектов в одну строку

```
String s = String.Format("On {0}, {1} is {2} years old.",  
    new DateTime(2021, 4, 22, 14, 35, 5), "Aidan", 9);
```

```
String s = String.Format("On {0:D}, {1} is {2:E} years old.",  
    new DateTime(2021, 4, 22, 14, 35, 5), "Aidan", 9);
```

Получение объекта посредством разбора строки

```
public static Int32 Parse(String s, NumberStyles style, IFormatProvider provider);
```

```
Int32 x = Int32.Parse(" 123", NumberStyles.None, null);
```

```
Int32 x = Int32.Parse(" 123", NumberStyles.AllowLeadingWhite, null);
```

```
Int32 x = Int32.Parse("1A", NumberStyles.HexNumber, null);
```

```
public static Int32 Parse(String s);  
// Передает информацию о региональных стандартах потока  
public static Int32 Parse(String s, NumberStyles style);  
// Передает NumberStyles.Integer в качестве параметра стиля  
public static Int32 Parse(String s, IFormatProvider provider)  
// Тот метод, о котором я уже рассказывал в этом разделе  
public static int Parse(String s, NumberStyles style,  
IFormatProvider provider);
```



Кодировки: преобразования между символами и байтами

Кодировки

- UTF-16
- UTF-8
- UTF-32
- UTF-7
- ASCII



Кодирование и декодирование

```
// Кодируемая строка
String s = "Hi there.";
// Получаем объект, производный от Encoding, который "умеет" выполнять
// кодирование и декодирование с использованием UTF-8
Encoding encodingUTF8 = Encoding.UTF8;
// Выполняем кодирование строки в массив байтов
Byte[] encodedBytes = encodingUTF8.GetBytes(s);
// Показываем значение закодированных байтов
Console.WriteLine("Encoded bytes: " +
    BitConverter.ToString(encodedBytes));
// Выполняем декодирование массива байтов обратно в строку
String decodedString = encodingUTF8.GetString(encodedBytes);
// Показываем декодированную строку
Console.WriteLine("Decoded string: " + decodedString);
```



Кодирование и декодирование потоков символов и байтов

Чтобы выполнить декодирование порции данных, следует получить ссылку на производный от `Encoding` объект и вызвать его метод `GetDecoder`

Тип, производный от `Encoding`, может служить для кодирования/декодирования без отслеживания состояния. Однако тип, производный от `Decoder`, можно использовать только для декодирования

Кодировка Base-64

```
// Получаем набор из 10 байт, сгенерированных случайным образом
Byte[] bytes = new Byte[10];
new Random().NextBytes(bytes);
// Отображаем байты
Console.WriteLine(BitConverter.ToString(bytes));
// Декодируем байты в строку в кодировке base-64 и выводим эту строку
String s = Convert.ToBase64String(bytes);
Console.WriteLine(s);
// Кодируем строку в кодировке base-64 обратно в байты и выводим их
bytes = Convert.FromBase64String(s);
Console.WriteLine(BitConverter.ToString(bytes));
```



Защищенные строки

Класс System.Security.SecureString

```
public static void Main()
{
    using (SecureString ss = new SecureString())
    {
        Console.WriteLine("Please enter password: ");
        while (true)
        {
            ConsoleKeyInfo cki = Console.ReadKey(true);
            if (cki.Key == ConsoleKey.Enter) break;
            // Присоединить символы пароля в конец SecureString
            ss.AppendChar(cki.KeyChar);
            Console.Write("*");
        }
        Console.WriteLine();
        // Пароль введен, отобразим его для демонстрационных целей
        DisplaySecureString(ss);
    }
}
```

```
// Этот метод небезопасен, потому что обращается к неуправляемой памяти
private unsafe static void DisplaySecureString(SecureString ss)
{
    Char* pc = null;
    try
    {
        // Дешифрование SecureString в буфер неуправляемой памяти
        pc = (Char*)Marshal.SecureStringToCoTaskMemUnicode(ss);
        // Доступ к буферу неуправляемой памяти,
        // который хранит дешифрованную версию SecureString
        for (Int32 index = 0; pc[index] != 0; index++)
            Console.Write(pc[index]);
    }
    finally
    {
        // Обеспечиваем обнуление и освобождение буфера неуправляемой памяти,
        // который хранит расшифрованные символы SecureString
        if (pc != null)
            Marshal.ZeroFreeCoTaskMemUnicode((IntPtr)pc);
    }
}
```



Интерполяция строк

```
int x = 4;  
Console.WriteLine($"A square has {x} sides"); // Выводит: A square has 4 sides
```

```
string s = $"255 in hex is { byte.MaxValue:X2}"; // X2 - шестнадцатеричное  
// значение из двух цифр  
// s получает значение "255 in hex is FF"
```

```
int x = 2;  
string s = @$"this spans {  
x} lines";
```



Регулярные выражения

```
string s = "Корабли лавировали лавировали да никак не вылавировали";  
Regex regex = new Regex(@"лавировал(\w*)");  
MatchCollection matches = regex.Matches(s);  
if (matches.Count > 0)  
{  
    foreach (Match match in matches)  
        Console.WriteLine(match.Value);  
}  
else  
{  
    Console.WriteLine("Совпадений не найдено");  
}
```



Параметр RegexOptions

- Compiled
- CultureInfoInvariant
- IgnoreCase
- IgnorePatternWhitespace
- Multiline
- RightToLeft
- Singleline

```
Regex regex = new Regex(@"лавироровал(\w*)", RegexOptions.IgnoreCase);
```

```
Regex regex = new Regex(@"лавироровал(\w*)", RegexOptions.Compiled | RegexOptions.IgnoreCase);
```



Замена и метод Replace

```
string s = "Привет из СПб! ";  
string pattern = @"\s+";  
string target = " ";  
Regex regex = new Regex(pattern);  
string result = regex.Replace(s, target);
```



Преобразование простых типов

- Приведение типа
- Использование типа Convert
- Использование интерфейса IConvertible

```
// Преобразование "число - символ" посредством приведения типов C#
Char c = (Char)65;
Console.WriteLine(c); // Выводится "A"
Int32 n = (Int32)c;
Console.WriteLine(n); // Выводится "65"
c = unchecked((Char)(65536 + 65));
Console.WriteLine(c); // Выводится "A"
```

```
// Преобразование "число - символ" с помощью типа Convert
c = Convert.ToChar(65);
Console.WriteLine(c); // Выводится "A"
n = Convert.ToInt32(c);
Console.WriteLine(n); // Выводится "65"
```

```
// Демонстрация проверки диапазона для Convert
try
{
    c = Convert.ToChar(70000); // Слишком много для 16 разрядов
    Console.WriteLine(c); // Этот вызов выполняться НЕ будет
}
catch (OverflowException)
{
    Console.WriteLine("Can't convert 70000 to a Char.");
}
```

```
// Преобразование "число - символ" с помощью интерфейса IConvertible
c = ((IConvertible)65).ToChar(null);
Console.WriteLine(c); // Выводится "A"
n = ((IConvertible)c).ToInt32(null);
Console.WriteLine(n); // Выводится "65"
```



Исключения и управление состоянием

Что такое «ИСКЛЮЧЕНИЕ»

```
public static void Transfer(Account from, Account to, Decimal amount)
{
    from -= amount;
    to += amount;
}
```

```
class Account
{
    public Account(Decimal amount)
    {
        Current = amount;
    }

    public Decimal Current { get; private set; }

    public static Account operator +(Account acc, Decimal amount) {
        return new Account(acc.Current + amount);
    }
    public static Account operator -(Account acc, Decimal amount) {
        return new Account(acc.Current - amount);
    }
}
```

```
static void Transfer(Account from, Account to, Decimal amount)
{
    if (from.Current - amount < 0)
        throw new Exception("Can not transfer such amount");

    from -= amount;
    to += amount;
}

static void Main(string[] args)
{
    try
    {
        Account a1 = new Account(10);
        Account a2 = new Account(20);
        Transfer(a2, a1, 100);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
        throw;
    }
}
```

Механика обработки исключений

```
private void SomeMethod()
{
    try
    {
        // Код, требующий корректного восстановления
        // или очистки ресурсов
    }
    catch (InvalidOperationException)
    {
        // Код восстановления работоспособности
        // после исключения InvalidOperationException
    }
    catch (IOException)
    {
        // Код восстановления работоспособности
        // после исключения IOException
    }
    catch
    {
        // Код восстановления работоспособности после остальных исключений.
        // После перехвата исключений их обычно генерируют повторно
        // Эта тема будет рассмотрена позже
        throw;
    }
    finally
    {
        // Здесь находится код, выполняющий очистку ресурсов
        // после операций, начатых в блоке try. Этот код
        // выполняется ВСЕГДА вне зависимости от наличия исключения
    }
    // Код, следующий за блоком finally, выполняется, если в блоке try
    // не генерировалось исключение или если исключение было перехвачено
    // блоком catch, а новое не генерировалось
}
```



Блок finally

```
FileStream fs = null;
try
{
    fs = new FileStream(pathname, FileMode.Open);
    // Обработка данных в файле
}
catch (IOException)
{
    // Код восстановления после исключения IOException
}
finally
{
    // Файл обязательно следует закрыть
    if (fs != null) fs.Close();
}
```



Класс System.Exception

- Message
- Data
- Source
- StackTrace
- TargetSite
- HelpLink
- InnerException
- HResult



Обнуление начальной точки

```
private void SomeMethod()  
{  
    try { ... }  
    catch (Exception e)  
    {  
        ...  
        throw e; // CLR считает, что исключение возникло тут  
    }  
}
```

```
private void SomeMethod()  
{  
    try { ... }  
    catch (Exception e)  
    {  
        ...  
        throw; // CLR не меняет информацию о начальной точке исключения.  
    }  
}
```



Продуктивность вместо надежности

```
public static void Transfer(Account from, Account to, Decimal amount)
{
    try { /* здесь ничего не делается */ }
    finally
    {
        from -= amount;
        to += amount;
    }
}
```

```
public static class SomeType
{
    private static Object s_myLockObject = new Object();
    public static void SomeMethod()
    {
        Monitor.Enter(s_myLockObject); // В случае исключения произойдет ли
                                         // блокировка? Если да, то этот режим
                                         // будет невозможно отключить!

        try
        {
            // Безопасная в отношении потоков операция
        }
        finally
        {
            Monitor.Exit(s_myLockObject);
        }
    }
    // ...
}
```

```
public static class SomeType
{
    private static Object s_myLockObject = new Object();
    public static void SomeMethod()
    {
        Boolean lockTaken = false; // Предполагаем, что блокировки нет
        try
        {
            // Это работает вне зависимости от наличия исключения!
            Monitor.Enter(s_myLockObject, ref lockTaken);
            // Потокобезопасная операция
        }
        finally
        {
            // Если режим блокировки включен, отключаем его
            if (lockTaken) Monitor.Exit(s_myLockObject);
        }
    }
    // ...
}
```


Приемы работы с исключениями

Активно используйте блоки finally

```
private void SomeMethod()
{
    // Открытие файла
    FileStream fs = new FileStream(@"C:\Data.bin ", FileMode.Open);
    try
    {
        // Вывод частного от деления 100 на первый байт файла
        Console.WriteLine(100 / fs.ReadByte());
    }
    finally
    {
        // В блоке finally размещается код очистки, гарантирующий
        // закрытие файла независимо от того, возникло исключение
        // (например, если первый байт файла равен 0) или нет
        fs.Close();
    }
}
```

```
private void SomeMethod()
{
    using (FileStream fs =
        new FileStream(@"C:\Data.bin", FileMode.Open))
    {
        // Вывод частного от деления 100 на первый байт файла
        Console.WriteLine(100 / fs.ReadByte());
    }
}
```



Не надо перехватывать все исключения

```
try {  
    // Попытка выполнить код, который, как считает программист,  
    // может привести к сбою...  
}  
catch (Exception) {  
    ...  
}
```



Восстановление после исключения

```
public String CalculateSpreadsheetCell(Int32 row, Int32 column)
{
    String result;
    try
    {
        result = /* Код для расчета значения ячейки электронной таблицы */
    }
    catch (DivideByZeroException)
    {
        result = "Нельзя отобразить значение: деление на ноль";
    }
    catch (OverflowException)
    {
        result = "Нельзя отобразить значение: оно слишком большое";
    }
    return result;
}
```



Отмена незавершенных операций при невозстановимых исключениях

```
public void SerializeObjectGraph(FileStream fs,
    IFormatter formatter, Object rootObj)
{
    // Сохранение текущей позиции в файле
    Int64 beforeSerialization = fs.Position;
    try
    {
        // Попытка сериализовать граф объекта и записать его в файл
        formatter.Serialize(fs, rootObj);
    }
    catch
    { // Перехват всех исключений
        // При ЛЮБОМ повреждении файл возвращается в нормальное состояние
        fs.Position = beforeSerialization;
        // Обрезаем файл
        fs.SetLength(fs.Position);
        // ПРИМЕЧАНИЕ: предыдущий код не помещен в блок finally,
        // так как сброс потока требуется только при сбое сериализации
        // Уведомляем вызывающий код о происходящем,
        // снова генерируя ТО ЖЕ САМОЕ исключение
        throw;
    }
}
```



Соккрытие деталей реализации для сохранения контракта

```
class PhoneBook
{
    private String m_pathname; // Путь к файлу с телефонами
                                // Выполнение других методов
    public String GetPhoneNumber(String name)
    {
        String phone;
        FileStream fs = null;
        try
        {
            fs = new FileStream(m_pathname, FileMode.Open);
            // Чтение переменной fs до обнаружения нужного имени
            phone = /* номер телефона найден */

            catch (FileNotFoundException e)
            {
                // Генерирование другого исключения, содержащего имя абонента,
                // с заданием исходного исключения в качестве внутреннего
                throw new NameNotFoundException(name, e);
            }
            catch (IOException e)
            {
                // Генерирование другого исключения, содержащего имя абонента,
                // с заданием исходного исключения в качестве внутреннего
                throw new NameNotFoundException(name, e);
            }
            finally
            {
                if (fs != null) fs.Close();
            }
            return phone;
        }
    }
}
```



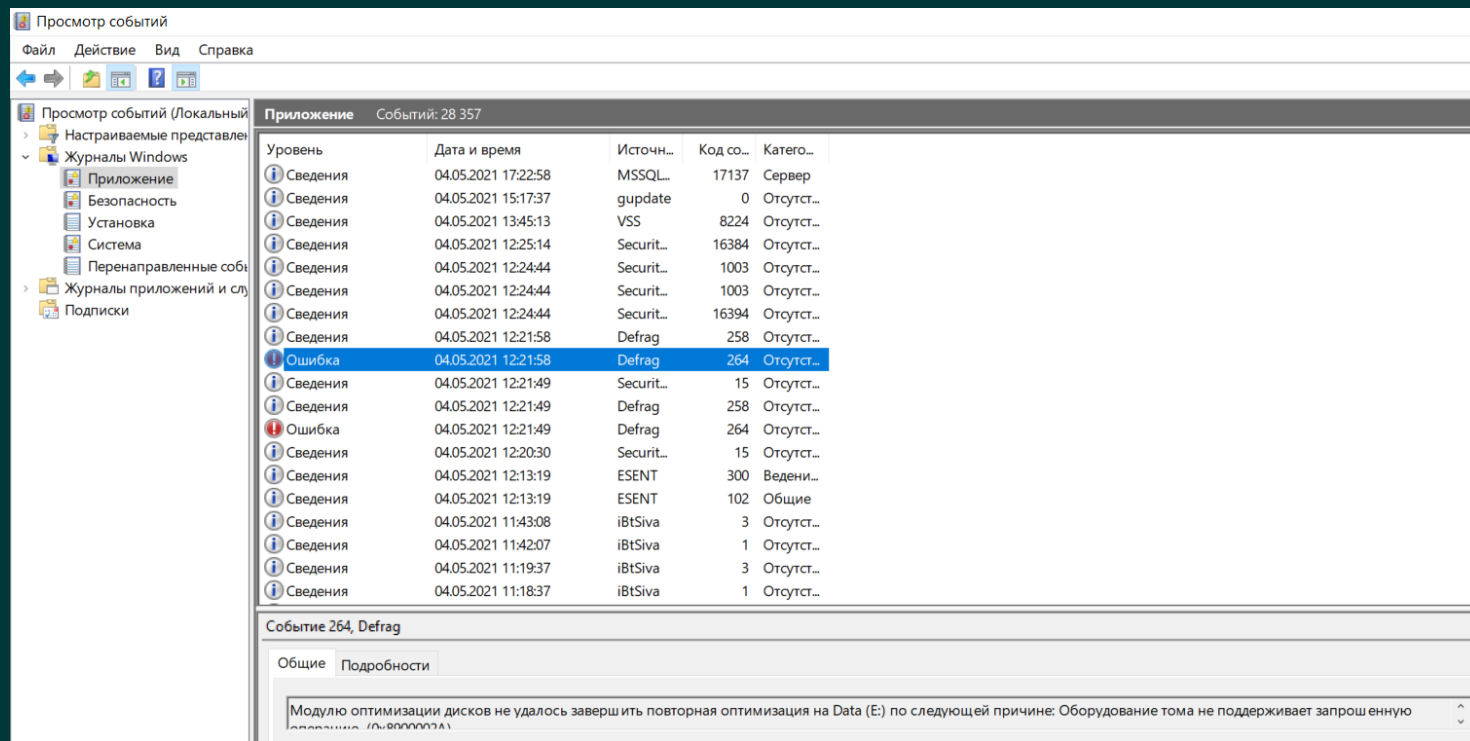
Генерирование после перехвата

```
private static void SomeMethod(String filename)
{
    try
    {
        // Какие-то операции
    }
    catch (IOException e)
    {
        // Добавление имени файла к объекту IOException
        e.Data.Add("Filename", filename);
        throw; // повторное генерирование того же исключения
    }
}
```



Необработанные исключения

Event Viewer: Windows Logs -> Application



Просмотр событий

Файл Действие Вид Справка

Просмотр событий (Локальный компьютер)

- Настраиваемые представления
- Журналы Windows
 - Приложение
 - Безопасность
 - Установка
 - Система
 - Перенаправленные события
- Журналы приложений и служб
- Подписки

Приложение Событий: 28 357

Уровень	Дата и время	Источн...	Код со...	Катего...
Сведения	04.05.2021 17:22:58	MSSQL...	17137	Сервер
Сведения	04.05.2021 15:17:37	gupdate	0	Отсутст...
Сведения	04.05.2021 13:45:13	VSS	8224	Отсутст...
Сведения	04.05.2021 12:25:14	Securit...	16384	Отсутст...
Сведения	04.05.2021 12:24:44	Securit...	1003	Отсутст...
Сведения	04.05.2021 12:24:44	Securit...	1003	Отсутст...
Сведения	04.05.2021 12:24:44	Securit...	16394	Отсутст...
Сведения	04.05.2021 12:21:58	Defrag	258	Отсутст...
Ошибка	04.05.2021 12:21:58	Defrag	264	Отсутст...
Сведения	04.05.2021 12:21:49	Securit...	15	Отсутст...
Сведения	04.05.2021 12:21:49	Defrag	258	Отсутст...
Ошибка	04.05.2021 12:21:49	Defrag	264	Отсутст...
Сведения	04.05.2021 12:20:30	Securit...	15	Отсутст...
Сведения	04.05.2021 12:13:19	ESENT	300	Ведени...
Сведения	04.05.2021 12:13:19	ESENT	102	Общие
Сведения	04.05.2021 11:43:08	iBtSiva	3	Отсутст...
Сведения	04.05.2021 11:42:07	iBtSiva	1	Отсутст...
Сведения	04.05.2021 11:19:37	iBtSiva	3	Отсутст...
Сведения	04.05.2021 11:18:37	iBtSiva	1	Отсутст...

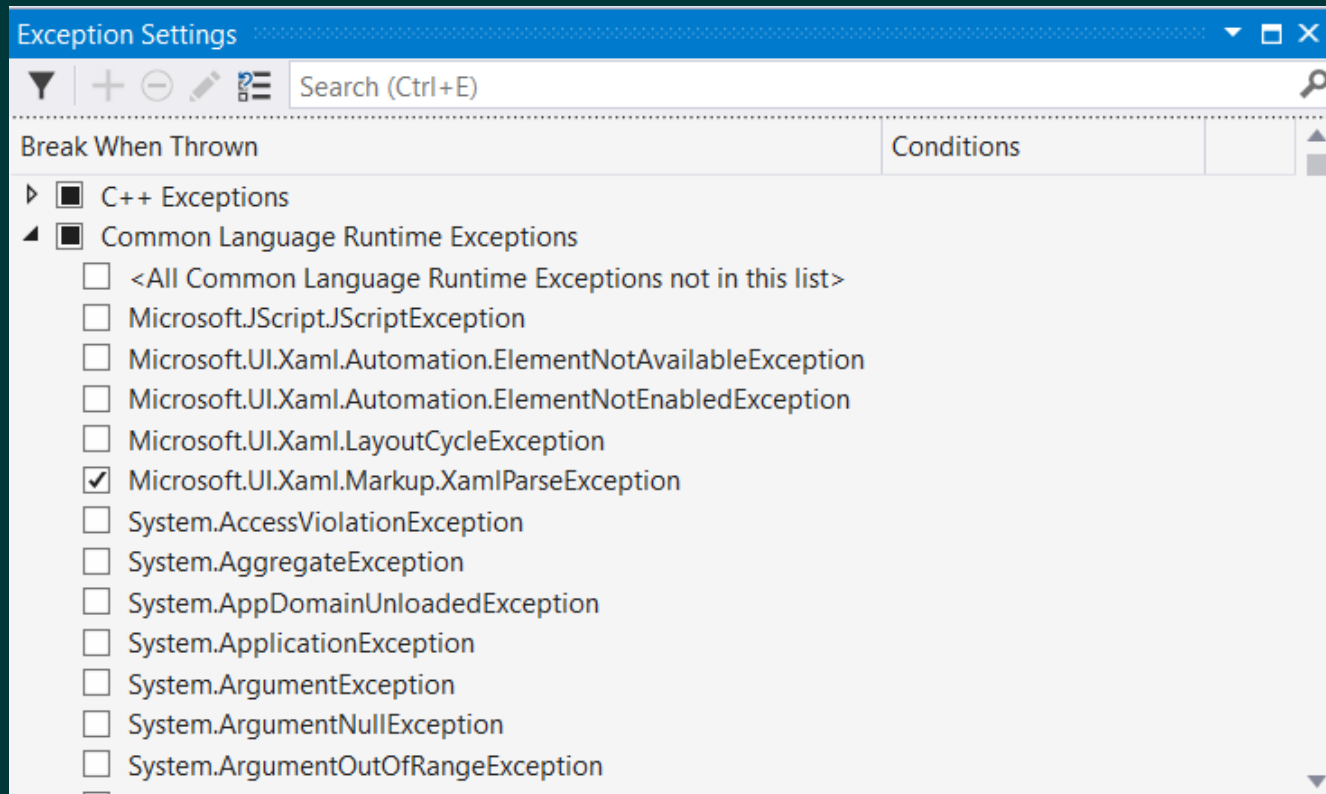
Событие 264, Defrag

Общие Подробности

Модулю оптимизации дисков не удалось завершить повторная оптимизация на Data (E:) по следующей причине: Оборудование тома не поддерживает запрошенную оптимизацию. (0x80000000)



Отладка исключений



Области ограниченного выполнения

Областью ограниченного выполнения (Constrained Execution Region, CER) называется фрагмент кода, который должен быть устойчивым к сбоям

```
private sealed class Type1
{
    static Type1()
    {
        // В случае исключения М не вызывается
        Console.WriteLine("Type1's static ctor called");
    }
    public static void M() { }
}

private static void Demo1()
{
    try
    {
        Console.WriteLine("In try");
    }
    finally
    {
        // Неявный вызов статического конструктора Type1
        Type1.M();
    }
}
```

In try
Type1's static ctor called

```
private static void Demo2()
{
    // Подготавливаем код в блоке finally
    RuntimeHelpers.PrepareConstrainedRegions(); // Пространство имен
                                                // System.Runtime.CompilerServices

    try
    {
        Console.WriteLine("In try");
    }
    finally
    {
        // Неявный вызов статического конструктора Type2
        Type2.M();
    }
}

public class Type2
{
    static Type2()
    {
        Console.WriteLine("Type2's static ctor called");
    }
    // Используем атрибут, определенный в пространстве имен
    // System.Runtime.ConstrainedExecution
    [ReliabilityContract(Consistency.WillNotCorruptState, Cer.Success)]
    public static void M() { }
}
```

Type2's static ctor called
In try



CLS-совместимые и CLS-несовместимые исключения

```
try
{
    // Внутри блока try помещают код, требующий корректного
    // восстановления работоспособности или очистки ресурсов
}
catch (Exception e)
{
    // До C# 2.0 этот блок перехватывал только CLS-совместимые исключения
    // В C# 2.0 этот блок научился перехватывать также
    // CLS-несовместимые исключения
    throw; // Повторная генерация перехваченного исключения
}
catch
{
    // Во всех версиях C# этот блок перехватывает
    // и совместимые, и несовместимые с CLS исключения
    throw; // Повторная генерация перехваченного исключения
}
```



ExceptionDispatchInfo

Представляет исключение, состояние которого регистрируется в определенной точке кода

Исключение может быть вызвано в другое время и, возможно, в другом потоке путем вызова метода `ExceptionDispatchInfo.Throw`

```
if (s_exceptions.Count == 1)
{
    ExceptionDispatchInfo dispatchInfo = ExceptionDispatchInfo.Capture(s_exceptions[0]);
    dispatchInfo.Throw();
}
else
{
    throw new AggregateException(s_exceptions);
}
```



Почему важно перехватывать все исключения

В любой точке входа программы нужно вставлять try-catch и перехватывать все исключения

```
private static int Main()
{
    try
    {
        Console.WriteLine("Console application started.");

        // Делаем что-то полезное
        return 0;
    }
    catch (Exception ex)
    {
        Console.WriteLine($"Exception occurred: {Environment.NewLine}{ex.Message}");
        return -1;
    }
    finally
    {
        Console.WriteLine("Console application stopped.");
        Console.WriteLine("Press any key to close this window...");
        Console.ReadKey();
    }
}
```

```
[HttpPost]
public async Task<ActionResult<ProcessingResponse>> PostRequestData(
    RequestData requestData)
{
    _logger.Info("Processing data request.");

    try
    {
        IServiceRequestProcessor requestProcessor = _serviceCreator.CreateRequestProcessor(
            requestData.ConfigurationXml.ServiceType
        );

        ProcessingResponse response = await requestProcessor.ProcessRequest(requestData);
        if (response.Metadata.ResultStatus != ServiceStatus.Ok)
        {
            return BadRequest(response);
        }
        return response;
    }
    catch (Exception ex)
    {
        _logger.Error(ex, "Exception occurred during data processing.");
    }
    return BadRequest(requestData);
}
```



Полезные ссылки



Thank you

veeam
ACADEMY

