

Исключения, рефлексия, модульное тестирование

Юрий Литвинов
yurii.litvinov@gmail.com

21.09.2017г

Бросание исключений

```
if (t == null)
{
    throw new NullPointerException();
}

throw new NullPointerException("Hello!");
```

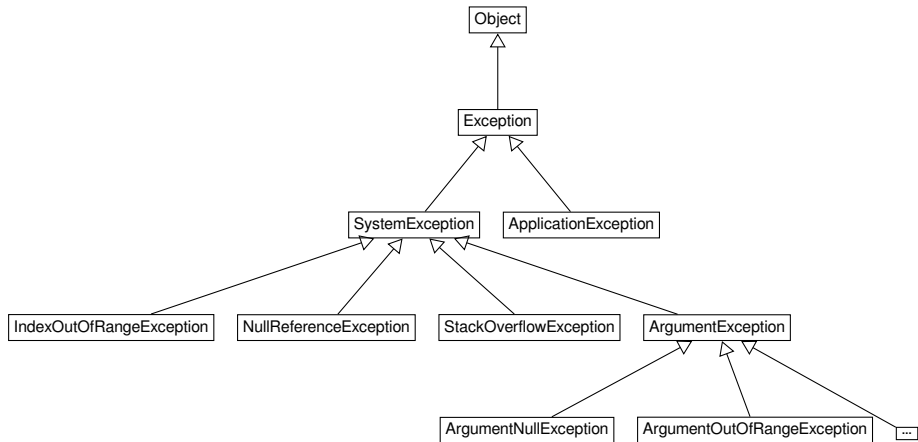
Обработка исключений

```
try {  
    // Код, который может бросать исключения  
} catch (Type1 id1) {  
    // Обработка исключений типа Type1  
} catch (Type2 id2) {  
    // Обработка исключений типа Type2  
} catch {  
    // Обработка всех оставшихся исключений  
} finally {  
    // Код, выполняющийся в любом случае  
}
```

Пример

```
private void ReadData(String pathname)
{
    FileStream fs = null;
    try {
        fs = new FileStream(pathname, FileMode.Open);
        // Делать что-то полезное
    }
    catch (IOException) {
        // Код восстановления после ошибки
    }
    finally {
        // Закрывать файл надо в любом случае
        if (fs != null) fs.Close();
    }
}
```

Иерархия классов-исключений



Свойства класса Exception

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

Перебрасывание исключений

```
try
{
    throw new Exception("Something is wrong");
}
catch (Exception e)
{
    Console.WriteLine("Caught Exception");
    throw; // "throw e;" тоже работает, но сбросит stack trace
}
```

Или

```
throw new Exception("Outer exception", e);
```

Свои классы-исключения

```
public class MyException : ApplicationException
{
    public MyException()
    {
    }

    public MyException(string message)
        : base(message)
    {
    }
}
```


Идеологически правильное объявление исключения

[Serializable]

```
public class MyException : Exception
{
    public MyException() { }
    public MyException(string message) : base(message) { }
    public MyException(string message, Exception inner)
        : base(message, inner) { }
    protected MyException(
        System.Runtime.Serialization.SerializationInfo info,
        System.Runtime.Serialization.StreamingContext context)
        : base(info, context) { }
}
```

“Интересные” классы-исключения

- ▶ Corrupted state exceptions (CSE) — не ловятся catch-ем
 - ▶ StackOverflowException
 - ▶ AccessViolationException
 - ▶ System.Runtime.InteropServices.SEHException
- ▶ FileLoadException, BadImageFormatException, InvalidProgramException, ...
- ▶ ThreadAbortException
- ▶ TypeInitializationException
- ▶ TargetInvocationException
- ▶ OutOfMemoryException
- ▶ Можно кидать null. При этом рантайм сам кидает NullReferenceException.

Environment.FailFast

```
try {  
    // Делать что-то полезное  
}  
catch (OutOfMemoryException e)  
{  
    Console.WriteLine("Закончилась память :(");  
    Environment.FailFast(  
        String.Format($"Out of Memory: {e.Message}"));  
}
```

Исключения, best practices

- ▶ Не бросать исключения без нужды
 - ▶ В нормальном режиме работы исключения бросаться не должны вообще
- ▶ Свои исключения наследовать от `System.Exception`
- ▶ Документировать все свои исключения, бросаемые методом
- ▶ Не ловить `Exception`, `SystemException`
 - ▶ Исключения, указывающие на ошибку в коде (например, `NullReferenceException`) уж точно не ловить
- ▶ По возможности кидать библиотечные исключения или их наследников:
 - ▶ `InvalidOperationException`
 - ▶ `ArgumentException`
- ▶ Имя класса должно заканчиваться на “Exception”
- ▶ Код должен быть безопасным с точки зрения исключений
 - ▶ Не забывать про `finally`
 - ▶ Или `using`, `lock`, `foreach`, которые тоже генерят `finally`

Code Contracts

Набор инструментов для контрактно-ориентированного программирования

- ▶ Предусловия — должны выполняться до исполнения метода
- ▶ Постусловия — должны выполняться после исполнения метода
- ▶ Инварианты — должны выполняться всегда, пока объект не исполняет метод
 - ▶ Внутри метода инвариант объекта может нарушаться
- ▶ Класс `System.Diagnostics.Contracts.Contract`
- ▶ Требуется серьёзной инструментальной поддержки, чтобы делать что-то интересное

Пример

```

public sealed class ShoppingCart {
    private List<Item> cart = new List<Item>();
    private Decimal totalCost = 0;
    public void AddItem(Item item) => AddItemHelper(cart, item, ref totalCost);

    private static void AddItemHelper(List<Item> cart, Item newItem, ref Decimal totalCost) {
        // Предусловия:
        Contract.Requires(newItem != null);
        Contract.Requires(Contract.ForAll(cart, s => s != newItem));
        // Постусловия:
        Contract.Ensures(Contract.Exists(cart, s => s == newItem));
        Contract.Ensures(totalCost >= Contract.OldValue(totalCost));
        Contract.EnsuresOnThrow<IOException>(totalCost == Contract.OldValue(totalCost));
        // Делаем полезную работу...
        cart.Add(newItem);
        totalCost += 1.00M;
    }

    [ContractInvariantMethod]
    private void ObjectInvariant() {
        Contract.Invariant(totalCost >= 0);
    }
}

```

Рефлексия

- ▶ Позволяет во время выполнения получать информацию о типах
 - ▶ И главное, создавать объекты этих типов и вызывать их методы
- ▶ Зачем:
 - ▶ Плагины
 - ▶ Анализаторы кода
 - ▶ Тестовые системы
 - ▶ ...
- ▶ Проблемы:
 - ▶ Медленно
 - ▶ Нет помощи от системы типов

Загрузка сборки

```
public class Assembly {  
    public static Assembly Load(AssemblyName assemblyRef);  
    public static Assembly Load(String assemblyString);  
    public static Assembly Load(byte[] rawAssembly)  
    public static Assembly LoadFrom(String path);  
    public static Assembly ReflectionOnlyLoad(String assemblyString);  
    public static Assembly ReflectionOnlyLoadFrom(String assemblyFile);  
}
```

например,

```
var a = Assembly.LoadFrom(@"http://example.com/ExampleAssembly.dll");
```

Выгружать сборки нельзя

Пример

Распечатать имена всех типов в сборке

```
using System;  
using System.Reflection;
```

```
public static class Program {  
    public static void Main() {  
        string dataAssembly = "System.Data, version=4.0.0.0, "  
            + "culture=neutral, PublicKeyToken=b77a5c561934e089";  
        LoadAssemblyAndShowPublicTypes(dataAssembly);  
    }  
  
    private static void LoadAssemblyAndShowPublicTypes(string assemblyId) {  
        var a = Assembly.Load(assemblyId);  
        foreach (Type t in a.ExportedTypes) {  
            Console.WriteLine(t.FullName);  
        }  
    }  
}
```

Создание экземпляра объекта

- ▶ `System.Activator.CreateInstance` — можно передавать тип или строку с именем типа
 - ▶ Версии со строкой возвращают `System.Runtime.Remoting.ObjectHandle`, надо вызвать `Unwrap()`
- ▶ `System.Activator.CreateInstanceFrom` — вызывает `LoadFrom` для сборки
- ▶ `System.Reflection.ConstructorInfo.Invoke` — просто вызов конструктора (несколько дольше писать, чем предыдущие варианты)
- ▶ Рефлексия ничего не знает о синонимах

Создание экземпляра типа-генерика

```
using System;
using System.Reflection;

internal sealed class Dictionary<TKey, TValue> { }

public static class Program {
    public static void Main() {
        Type openType = typeof(Dictionary<,>);
        Type closedType = openType.MakeGenericType(
            typeof(String), typeof(Int32));
        Object o = Activator.CreateInstance(closedType);
        Console.WriteLine(o.GetType());
    }
}
```

Пример: как сделать свою плагинную систему

- ▶ Сделать отдельную сборку с описанием интерфейса плагина и типов данных, которые он использует
 - ▶ Менять её будет очень проблематично
- ▶ Сделать “ядро системы” — отдельную сборку, ссылающуюся на сборку с интерфейсом плагина
- ▶ Делать набор плагинов, ссылающихся на сборку с интерфейсом плагина и реализующих его

Пример: интерфейс плагина

```
namespace MyCoolSystem.SDK {  
    public interface IAddIn {  
        string DoSomething(int x);  
    }  
}
```

Пример: плагины

```
using MyCoolSystem.SDK;
```

```
public sealed class AddInA : IAddIn {  
    public String DoSomething(int x) {  
        return "AddInA: " + x.ToString();  
    }  
}
```

```
public sealed class AddInB : IAddIn {  
    public String DoSomething(int x) {  
        return "AddInB: " + (x * 2).ToString();  
    }  
}
```

Пример: ядро системы

```
public static class Program
```

```
{
```

```
    public static void Main()
```

```
{
```

```
    string addInDir = Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
```

```
    var addInAssemblies = Directory.EnumerateFiles(addInDir, "*.dll");
```

```
    var addInTypes =
```

```
        addInAssemblies.Select(Assembly.Load)
```

```
        .SelectMany(a => a.ExportedTypes)
```

```
        .Where(t => t.IsClass
```

```
            && typeof(IAddIn).GetTypeInfo().IsAssignableFrom(t.GetTypeInfo()));
```

```
    foreach (Type t in addInTypes)
```

```
{
```

```
    var addIn = (IAddIn)Activator.CreateInstance(t);
```

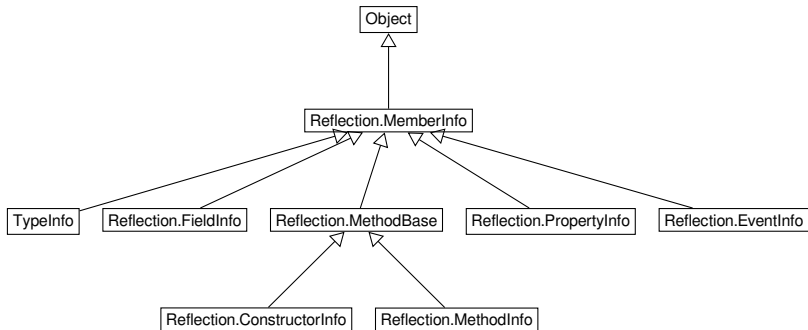
```
    Console.WriteLine(addIn.DoSomething(5));
```

```
}
```

```
}
```

```
}
```

Информация о типах



Пример: распечатать информацию о полях и методах

```

using System;
using System.Reflection;

public static class Program {
    public static void Main() {
        Assembly[] assemblies = AppDomain.CurrentDomain.GetAssemblies();
        foreach (Assembly a in assemblies) {
            Console.WriteLine($"Assembly: {a}");
            foreach (Type t in a.ExportedTypes) {
                Console.WriteLine($"Type: {t}");
                foreach (MemberInfo mi in t.GetTypeInfo().DeclaredMembers) {
                    var typeName = string.Empty;
                    if (mi is FieldInfo) typeName = "FieldInfo";
                    if (mi is MethodInfo) typeName = "MethodInfo";
                    if (mi is ConstructorInfo) typeName = "ConstructorInfo";
                    Console.WriteLine($" {typeName}: {mi}");
                }
            }
        }
    }
}

```

Полезные свойства MemberInfo

- ▶ Name (string) — имя члена класса
- ▶ DeclaringType (Type) — тип
- ▶ Module (Module) — модуль, в котором он объявлен
- ▶ CustomAttributes (IEnumerable<CustomAttributeData>) — коллекция атрибутов, соответствующих этому члену класса
 - ▶ Пример — модульные тесты

Как что-нибудь сделать с MemberInfo

- ▶ GetValue и SetValue для FieldInfo и PropertyInfo
- ▶ Invoke для ConstructorInfo и MethodInfo
- ▶ AddEventHandler и RemoveEventHandler для EventInfo

Пример: создать класс и вызвать его метод

```
using System;
using System.Reflection;
using System.Linq;
```

```
internal sealed class SomeType {
    public SomeType(int test) { }
    private int DoSomething(int x) => x * 2;
}
```

```
public static class Program {
    public static void Main() {
        Type t = typeof(SomeType);
        Type ctorArgument = Type.GetType("System.Int32");
        ConstructorInfo ctor = t.GetTypeInfo().DeclaredConstructors.First(
            c => c.GetParameters()[0].ParameterType == ctorArgument);
        Object[] args = { 12 };
        Object obj = ctor.Invoke(args);
        MethodInfo mi = obj.GetType().GetTypeInfo().GetDeclaredMethod("DoSomething");
        int result = (int)mi.Invoke(obj, new object[] { 3 });
        Console.WriteLine($"{result} = {result}");
    }
}
```

Модульное тестирование

- ▶ Помогает искать ошибки
 - ▶ Особо эффективно, если налажен процесс Continuous Integration
- ▶ Облегчает изменение программы, рефакторинг
 - ▶ Но несколько замедляет процесс, тесты тоже требуют сопровождения
- ▶ Тесты — документация к коду
- ▶ Тесты помогают улучшить архитектуру, спагетти-код не протестировать

Популярные библиотеки

- ▶ NUnit
 - ▶ Отдельный пакет
 - ▶ Интегрируется в IDE расширениями
- ▶ Microsoft Unit Test Framework
 - ▶ Работает прямо из коробки в Visual Studio, но требует некоторой возни, если нет
- ▶ XUnit
- ▶ MbUnit

Best practices

- ▶ Независимость тестов
 - ▶ Желательно, чтобы поломка одного куска функциональности ломала один тест
- ▶ Тесты должны работать быстро
 - ▶ И запускаться после каждой сборки
 - ▶ Continuous Integration!
- ▶ Тестов должно быть много
 - ▶ Следить за Code coverage, который многие инструменты умеют считать по тестовому прогону
- ▶ Каждый тест должен проверять конкретный тестовый сценарий
 - ▶ Никаких try-catch внутри теста
 - ▶ Атрибут ExpectedException для исключений
 - ▶ `[ExpectedException(typeof(NullReferenceException))]`
- ▶ Test-driven development

Mock-объекты

- ▶ Объекты-заглушки, симулирующие поведение реальных объектов и контролирующие обращения к своим методам
 - ▶ Как правило, такие объекты создаются с помощью библиотек
- ▶ Используются, когда реальные объекты использовать
 - ▶ Слишком долго
 - ▶ Слишком опасно
 - ▶ Слишком трудно
 - ▶ Для добавления детерминизма в тестовый сценарий
 - ▶ Пока реального объекта ещё нет
 - ▶ Для изоляции тестируемого объекта
- ▶ Для mock-объекта требуется, чтобы был интерфейс, который он мог бы реализовать, и какой-то механизм внедрения объекта
 - ▶ Паттерны “Dependency Injection”, “Стратегия”

Популярные библиотеки

- ▶ NSubstitute
- ▶ Moq
- ▶ Rhino Mocks
- ▶ ...

NSubstitute

Легковесная библиотека, очень удобно описывать поведение объектов:

```
var stackStub = Substitute.For<IStack>();  
stackStub.IsEmpty().Returns(false);  
stackStub.Pop().Returns(1);
```

```
Assert.AreEqual(1, stackStub.Pop());
```

```
stackStub.Received().Pop();  
stackStub.DidNotReceive().Push(Arg.Any<int>());
```

Moq

Активно использует лямбда-функции и LINQ:

```
var stackMock = new Mock<IStack>();  
stackMock.Setup(st => st.IsEmpty()).Returns(false);  
stackMock.Setup(st => st.Pop()).Returns(1);
```

```
var stack = stackMock.Object;  
stack.Push(1);  
stack.Pop();
```

```
stackMock.Verify(st => st.IsEmpty(), Times.Never);  
stackMock.Verify(st => st.Pop(), Times.Exactly(1));  
stackMock.Verify(st => st.Push(It.IsAny<int>()), Times.Exactly(1));
```

Moq (2)

Или так (LINQ-магия):

```
var stack = Mock.Of<IStack>(
    st => st.IsEmpty() == false && st.Pop() == 1);
```

```
stack.Push(1);
stack.Pop();
```

```
Mock.Get(stack).Verify(st => st.Push(It.IsAny<int>()), Times.Exactly(1));
Mock.Get(stack).Verify(st => st.Pop(), Times.Exactly(1));
Mock.Get(stack).Verify(st => st.IsEmpty(), Times.Never);
```

Rhino Mocks

```
var stack = MockRepository.Mock<IStack>();  
stack.Expect(st => st.Push(1));  
stack.Expect(st => st.Pop()).Returns(() => 1);  
  
stack.Push(1);  
Assert.AreEqual(1, stack.Pop());  
  
stack.VerifyAllExpectations();
```