Лабараторная работа №1

Разработать библиотеку, позволюющую производить замеры производительности приложения. Внешний интерфейс представляет собой статический класс Tracerсо следующими методами:

* Start – начинается очередной цикл профилирования. Без вызова этого метода реальной работы кода по замерам производительности встроенного в приложение не происходит
* Stop – заканчивается очередной цикл профилирования. В результате выполнения этого метода все данные полученные между Startи End возвращаются и могут быть использованы в дальнейшем для генерации отчета.
* BeginTrace – метод, который начинает замер времени в определенной строчке кода. Данный метод встраивается в код приложения. Вызов этого метода не имеет никакого эффекта, если не был вызван Start.
* EndTrace – метод, который заканчивает замер времени и должен идти после очередного вызова BeginTrace. Данный метод сохраняет результаты замера времени. Вызов этого метода не имеет никакого эффекта, если не был вызван Start.

Пары методов BeginTrace и EndTrace являются тем местом, которое ответственно за сбор необходимой информации о времени обработки метода. В результате вызова EndTrace должна быть получена следующая информация:

* время прошедшее между вызовом BeginTrace и EndTrace
* имя профилируемого метода
* название класса. В котором находитя профилируемый метод
* название модуля, в котором находится профилируемый класс.

Помимо этого методы BeginTrace и EndTrace могут иметь вложенную структуру, что отражается в результатах, как цепочка вложенных методов с соответствующими результатами по времени нахождения в методе. Любые затраты на генерацию результатов должны оказывать наименьшее влияние на время, учтенное как затраченное при профилировании.

Методы Start и Stop не могут быть вложенными.

После вызова метода Stop полученная информация о цикле профилирования должна быть выведена на экран консоли в виде строки, а так же отформатирована в виде xml файла и сохранена на жестком диске.

Для профилирования необходимо написать простое консольное приложение, в начале выполнения которого вызывается метод Start, а в конце метод Stop.

Между этими вызовами происходят произвольные вызовы BeginTrace, EndTrace, которые в том числе имеют вложенную структуру, а так же находятся в разных методах.

**Дополнительно:** Добавить поддержку многопоточности. То есть методы BeginTrace и EndTrace сохраняют еще и информацию о потоке, в котором они были вызваны, что отражается на отчете. Информацией о потоке является уникальный идентификатор потока, который присваивается средой .NETпри его создании.

Примеры консольного приложенияи результатов вывода в xmlс учетом многопоточности приведены ниже. Если Tracer будет разработан только с учетом одного потока, то xmlбудет иметь схожую структуру, но не содержать информации о потоке.

namespaceTracer.TestApp

{

classProgram

{

staticvoid Main(string[] args)

{

Client.Tracer.Start();

Client.Tracer.BeginTrace();

Client.Tracer.EndTrace();

Client.Tracer.BeginTrace();

Thread.Sleep(100);

varbgThread = newThread(ThreadStart);

bgThread.Start();

OtherMethod();

Mine();

Client.Tracer.EndTrace();

bgThread.Join();

varwholeResult = Client.Tracer.Stop();

// Здесь должен находится код по форматированному выводу результатов.

Console.ReadKey();

}

staticvoid Mine()

{

Console.WriteLine("test");

}

staticvoidThreadStart()

{

OtherMethod();

OneMoreMethod();

}

staticvoidOtherMethod()

{

Client.Tracer.BeginTrace();

Thread.Sleep(105);

OneMoreMethod();

Client.Tracer.EndTrace();

}

staticvoidOneMoreMethod()

{

Client.Tracer.BeginTrace();

Client.Tracer.EndTrace();

}

}

}

Пример кода для работы с утилитой Tracer

<?xml version="1.0" encoding="utf-8"?>

<tracing>

<thread id="10">

<call>

<method name="Main" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="1" />

</call>

<call>

<method name="Main" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="221">

<nestedCalls>

<call>

<method name="OtherMethod" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="105">

<nestedCalls>

<call>

<method name="OneMoreMethod" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="0" />

</call>

</nestedCalls>

</method>

</call>

</nestedCalls>

</method>

</call>

</thread>

<thread id="11">

<call>

<method name="OtherMethod" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="105">

<nestedCalls>

<call>

<method name="OneMoreMethod" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="0" />

</call>

</nestedCalls>

</method>

</call>

<call>

<method name="OneMoreMethod" module="Tracer.TestApp.exe" class="Tracer.TestApp.Program" elapsedMilliseconds="0" />

</call>

</thread>

</tracing>

Пример результирующего xml файла для консольного приложения из примера выше