Работа с AggregateException

Введение

Известно, что инфраструктура PLINQ, класс *Parallel* и объекты *Task* автоматически маршализируют исключения потребителю. Чтобы понять, почему это важно, рассмотрим показанный ниже запрос LINQ, который на первой итерации генерирует исключение *DivideByZeroException*:

```
try
{
  var query = from i in Enumerable.Range(0, 1000000)
              select 100 / i;
    …
}
catch (DivideByZeroException)
{
    …
}
```

Если запросить у инфраструктуры PLINQ распараллеливание такого запроса, и она проигнорирует обработку исключений, то вполне возможно, что исключение *DivideByZeroException* сгенерируется в *отдельном потоке*, пропустив ваш блок *catch* и вызвав аварийное завершение приложения.

Поэтому исключения автоматически перехватываются и повторно генерируются для вызывающего потока. Но, к сожалению, дело не сводится просто к перехвату *DivideByZeroException*. Поскольку параллельные библиотеки задействуют множество потоков, вполне возможна одновременная генерация двух и более исключений. Чтобы обеспечить получение сведений обо всех исключениях, по указанной причине исключения помещаются в контейнер *AggregateException*, свойство *InnerExceptions* которого содержит каждое из перехваченных исключений:

```
try
{
  var query = from i in ParallelEnumerable.Range(0, 1000000)
              select 100 / i;
    // Выполнить перечисление результатов запроса
    …
}
catch (AggregateException aex)
{
  foreach (Exception ex in aex.InnerExceptions)
    Console.WriteLine(ex.Message);
}
```

- Как инфраструктура PLINQ, так и класс *Parallel* при обнаружении первого исключения заканчивают выполнение запроса или цикла, не обрабатывая любые последующие элементы либо итерации тела цикла. Однако до завершения текущей итерации цикла могут быть сгенерированы дополнительные исключения. Первое возникшее исключение в *AggregateException* доступно через свойство *InnerException*.

Flatten и Handle

Класс *AggregateException* предоставляет пару методов для упрощения обработки исключений: *Flatten* и *Handle*.

Flatten

Объекты *AggregateException* довольно часто будут содержать другие объекты *AggregateException*. Пример, когда подобное может произойти — ситуация, при которой дочерняя задача генерирует исключение. Чтобы упростить обработку, можно устранить любой уровень вложения, вызвав метод *Flatten*. Этот метод возвращает новый объект *AggregateException* с обычным плоским списком внутренних исключений:

```
catch (AggregateException aex)
{
  foreach (Exception ex in aex.Flatten().InnerExceptions)
    myLogWriter.LogException(ex);
}
```

Handle

Иногда полезно перехватывать исключения только специфических типов, а исключения других типов генерировать повторно. Метод Handle класса *AggregateException* предлагает удобное сокращение. Он принимает предикат исключений, который будет запускаться на каждом внутреннем исключении:

```
public void Handle (Func<Exception, bool> predicate)
```

Если предикат возвращает *true*, то считается, что исключение “обработано”. После того, как делегат запустится на всех исключениях, произойдет следующее:

- если все исключения были “обработаны” (делегат возвратил *true*), то исключение не генерируется повторно;
- если были исключения, для которых делегат возвратил *false* (“необработанные”), то строится новый объект *AggregateException*, содержащий такие исключения, и затем он генерируется повторно.

Например, приведенный далее код в конечном итоге повторно генерирует другой объект *AggregateException*, который содержит одиночное исключение *NullReferenceException*:

```
var parent = Task.Factory.StartNew(() =>
{
  // Мы сгенерируем 3 исключения сразу, используя 3 дочерние задачи:
  int[] numbers = { 0 };
  var childFactory = new TaskFactory
      (TaskCreationOptions.AttachedToParent,
       TaskContinuationOptions.None);
  childFactory.StartNew(() => 5 / numbers[0]);  // Деление на ноль
  childFactory.StartNew(()->numbers[1]);        // Выход индекса за
                                                // допустимые пределы
  childFactory.StartNew(() => { throw null; }); // Ссылка null
});
try { parent.Wait(); }
catch (AggregateException aex)
{
  aex.Flatten().Handle(ex => // Обратите внимание, что по-прежнему
                             // нужно вызывать Flatten
  {
```

```
    if (ex is DivideByZeroException)
    {
      Console.WriteLine("Divide by zero"); // Деление на ноль
      return true;        // Это исключение
                          // "обработано"
    }
    if (ex is IndexOutOfRangeException)
    {
      Console.WriteLine("Index out of range"); // Выход индекса
                                               // за допустимые пределы
      return true; // Это исключение
                   // "обработано"
    }
    return false; // Все остальные исключения будут
                  // сгенерированы повторно
  });
}
```