Task Parallel Library (TPL)
Klasse Task – Teil 2

Ingo Köster

Diplom Informatiker (FH)

Ausnahmebehandlung in Tasks

› Unbehandelte Ausnahmen, welche in einem Task auftreten, werden an den aufrufenden Thread zurückgeliefert

› Die Ausnahmen werden bei der Verwendung der Methoden `Task.Wait` oder `Task<T>.Wait` zurückgeliefert oder beim Abfragen der Eigenschaft `Result` des Tasks

› Die Methoden `Wait` und `Result` können zum Abfangen der Ausnahme(n) einen `try-catch` Block verwenden

Ausnahmebehandlung in Tasks

- Ein Task kann mehrere Ausnahmen zurückliefern
  - Auch verschiedene Typen
- Ein aufrufender Thread wartet ggf. auf mehrere Tasks
- Um alle Ausnahmen behandeln zu können, werden alle Ausnahmen in einem Objekt vom Typ `AggregateException` gesammelt

Ausnahmebehandlung in Tasks

- `AggregateException` besitzt die Eigenschaft `InnerExceptions`, welche alle aufgetretenen Ausnahmen enthält
- Diese können behandelt oder weitergeworfen werden
- Wird nur eine Ausnahme ausgelöst, wird diese ebenfalls in einer `AggregateException` gespeichert

Ausnahmebehandlungs-Beispiel mit `Wait`

```
Task einTask = Task.Factory.StartNew(() =>
{
 throw new Exception("Fehlertext");
});

try
{
 einTask.Wait();        // Hier tritt die Exception auf
}
catch (AggregateException ae)
{
 foreach (var e in ae.InnerExceptions) {
  Console.WriteLine(e.Message);
 }
}
```

Ausnahmebehandlungs-Beispiel mit `Result`

```
int i = 0;
var task = Task<int>.Run(() => 1 / i);

try
{
 Console.WriteLine(task.Result);  // Hier tritt die Exception auf
}
catch (AggregateException ex)
{
 if (ex.InnerException is DivideByZeroException)
  Console.WriteLine("Division by 0");
 else
  Console.WriteLine(ex);
}
```

Hinweis aus der MSDN

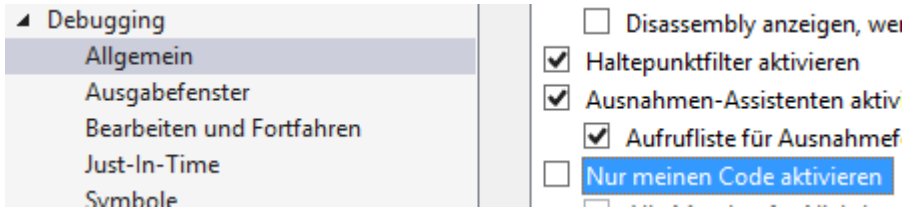
› Wenn "Nur eigenen Code" aktiviert ist, unterbricht Visual Studio die Ausführung in einigen Fällen in der Zeile, die die Ausnahme auslöst, und eine Fehlermeldung zu einer nicht vom Benutzercode behandelten Ausnahme wird angezeigt

› Um zu verhindern, dass Visual Studio die Ausführung beim ersten Fehler unterbricht, deaktivieren Sie das Kontrollkästchen

Nur meinen Code aktivieren

› unter

Extras -> Optionen -> Debugging -> Allgemein



Abbrechen eines Task

Abbrechen eines Task

- Innerhalb einer parallelen Operation kann diese mittels `return` abgebrochen werden

- Um einen Task von außen abbrechen zu können, ist ein Objekt vom Typ `CancellationTokenSource` notwendig
  - Das Objekt signalisiert einem Task, dass er seine Ausführung beenden soll

- Dem `CancellationTokenSource`-Objekt ist ein `CancellationToken`-Objekt zugeordnet, welches für die Weitergabe der Abbruchbenachrichtigung verantwortlich ist

Abbrechen eines Task

- Der Abbruch einer parallelen Ausführung von außen wird mit der Methode `Cancel` des `CancellationTokenSource`-Objekts eingeleitet
- Das Token gibt mit der booleschen Eigenschaft `IsCancellationRequested` Auskunft darüber, ob ein Abbruch gewünscht ist
- Ist der Wert `true`, kann der Task z.B. mit `return` beendet werden
- Oder den Task mit der Auslösung der Exception `OperationCanceledException` beenden
  - Kann durch den Aufruf der Methode `ThrowIfCancellationRequested` des Token-Objekt ausgelöst werden

Abbrechen eines Task

› Task „normal“ beenden

› Es ist nicht feststellbar, dass der Task abgebrochen wurde
  › Führt ggf. zu Problemen bei `ContinueWith`

```
if (token.IsCancellationRequested) { return; }
```

› Alternativ den Task mit einer Exception beenden

```
token.ThrowIfCancellationRequested();
```

Task starten und abbrechen

```
// Token anlegen
CancellationTokenSource cts = new CancellationTokenSource();

// Task starten
Task einTask = Task.Factory.StartNew(TaskMethode, cts.Token); //
Token übergeben
…
cts.Cancel(); // Task abbrechen

try { einTask.Wait(); } // Auf Task warten
catch (AggregateException e)
{ … } // Ausnahme bei Abbruch
```

Im Task auf Abbruch reagieren

```
void TaskMethode(object tokenObjekt) {
  Thread.Sleep(1000);
  CancellationToken token = (CancellationToken)tokenObjekt;

  while(true)
  {
    // Task abbrechen bei Aufruf von Cancel()
    token.ThrowIfCancellationRequested();
  }
 }
```

Token bei Lambda-Ausdruck nicht übergeben

```
CancellationTokenSource cts = new CancellationTokenSource();

CancellationToken token = cts.Token;

Task t = Task.Run(() =>
{
  …
  token.ThrowIfCancellationRequested();
  …
}); // hier kein Token übergeben!
```