Task Parallel Library (TPL)
Klasse Task – Teil 1

Ingo Köster

Diplom Informatiker (FH)

Klasse Task

› Threads müssen nicht vom Entwickler erstellt werden

› Für eine Aufgabenstellung wird ein Task definiert

› Das Erstellen der Threads erfolgt automatisch zur Laufzeit

Was ist ein Task?

- Im Kern ist ein Task eine Datenstruktur, die den eventuellen Abschluss einer asynchronen Operation darstellt
- Es wird ein Task erstellt, um eine Operation darzustellen
- Wenn die Operation des Tasks abgeschlossen ist, werden die Ergebnisse im Task gespeichert

Klasse Task

- Objekt vom Typ `Task` erstellen
- Übergebenes Objekt ist vom Typ `Action`
  - Action ist ein generisches Delegat
- Methode oder Lambda-Ausdruck wird als Hintergrund-Thread gestartet
  - D.h. die Tasks werden beendet, wenn das Hauptprogramm endet

Task starten - Verzögert

› Task anlegen und später starten

```
Task task1 = new Task(TaskMethode);
task1.Start();
…
static void TaskMethode()
{
  …
}
```

Task starten - Sofort

› Task anlegen und sofort starten

```
Task task2 = Task.Factory.StartNew(TaskMethode);
…
static void TaskMethode()
{
 …
}
```

› Auch mittels `Task.Run(TaskMethode)` möglich

Task starten – Lambda Ausdruck

› Task als Lambda-Ausdruck anlegen und sofort starten

```
Task task3 = Task.Factory.StartNew(() =>
{
  Console.WriteLine("Ein Lambda-Task wird ausgeführt ...");
});
```

`() => ...` bedeutet kein Übergabeparameter

Task starten – Übergabeparameter

› Task mit Übergabeparameter anlegen und sofort starten

```
string text = "Hallo Welt";
Task task4 = Task.Factory.StartNew(Ausgabe, text);
…
static void Ausgabe(object parameter)
{
 string s = (string)parameter;
 …
}
```

› Die Methode `Task.Run` erlaubt keine Übergabeparameter!

Task starten – Übergabeparameter

› Task mit Lambda-Ausdruck und Übergabeparameter

› Erst Lambda-Ausdruck angeben, dann Übergabeparameter

```
Task task5 = Task.Factory.StartNew(parameter =>
{
   string s = (string)parameter;
   Console.WriteLine("Lambda: {0}", s);
}
,text);
```

Auf Beendigung von Tasks warten

- Auf Beendigung eines einzelnen Task warten
  - Über die Objekt-Referenz
  - Beispiel: `task1.Wait();`

- Auf mehrere Tasks warten
  - Statische Methode der Klasse Task
  - Beispiel: `Task.WaitAll(task1, task2, task3, task4, task5);`

  - Alternative: `Task.WaitAny` wartet bis ein Task fertig ist
  - `WaitAny` liefert den Index des fertigen Task
  - Überladung der Methoden mit einem Task-Array möglich

Tasks mit Rückgabewerten

› Generischer Task mit Rückgabewert

```
int startWert = 14;
Task<int> task = Task<int>.Factory.StartNew(parameter =>
{
  int einWert = (int)parameter;
  Thread.Sleep(3000);
  return einWert * einWert;
}, startWert);
```

Tasks mit Rückgabewerten

- Abfragen des Rückgabewertes über die Eigenschaft `Result`
- `Result` ist typisiert (durch `Task<int> …`)
- Blockiert bis `Result` ein Ergebnis enthält
- Beispiel:
  - `Console.WriteLine("Resultat: {0}", task.Result);`

Rückgabewerte mit Task.Run

```
› Task<int> task = Task.Run(() => 1);
› Console.WriteLine(task.Result); // 1
```