



Was dir Trivialbeispiele in Async and Await nicht sagen!

Marcus Kimpenhaus und Martin Möllenbeck

Agenda

- Warum asynchrone Programmierung?
- Wie hat man dies bisher gelöst?
- „Async/await“ – das neue Pattern (TAP).
- Pragmatische Design Pattern für „async/await“ im Entwickler-Alltag.



Warum asynchrone Programmierung?

- Responsive UI (Fast & Fluid)
- Verbesserung des Datendurchsatz
- Optimierung des Thread-Pooling .NET Runtime
 - Beispiel: Web-Server
- Typische Szenarien
 - Web-Request's, I/O, Image-Processing (LRT), Database-Request's

Beispiel: Das Cafe



Wie hat man dies bisher gelöst? (1)

- APM: Asynchronous Programming Model
 - **BEGIN / END** Methoden / **IAsyncResult**
 - Callback-Handle (oder Lambda)

Weiterführende Links

- MSDN-Dokumentation: <http://tinyurl.com/osjwqtqe>



Wie hat man dies bisher gelöst? (1-Demo)

```
1 reference
class Program
{
    0 references
    static void Main(string[] args)
    {
        var program = new Program();
        program.LookupHostName();

        Console.WriteLine("Waiting for response...{0}", Environment.NewLine);
        Console.ReadKey();
    }

    1 reference
    private void LookupHostName()
    {
        var context = string.Format("Request started at: {0}", DateTime.Now.ToString("O"));
        Dns.BeginGetHostAddresses("google.com", this.OnHostNameResolved, context);
    }

    1 reference
    private void OnHostNameResolved(IAsyncResult result)
    {
        var context = result.AsyncState;

        Console.WriteLine(context);
        Console.WriteLine("Request finished at {0}", DateTime.Now.ToString("O"));

        var addresses = Dns.EndGetHostAddresses(result);

        Console.WriteLine("{0}IP-Adresses for google.com:", Environment.NewLine);

        foreach (var address in addresses)
        {
            Console.WriteLine(address);
        }
    }
}
```



Wie hat man dies bisher gelöst? (2)

- EAP: Event-based Asynchronous Pattern
 - Method **Async / Completed**
 - Event-Binding

Weiterführende Links

- MSDN-Dokumentation: <http://tinyurl.com/o6ump3u>



Wie hat man dies bisher gelöst? (2-Demo)

```
1 reference
class Program
{
    0 references
    static void Main(string[] args)
    {
        var program = new Program();
        program.DumpWebPage();

        Console.WriteLine("Waiting for response...{0}", Environment.NewLine);
        Console.ReadKey();
    }

    1 reference
    private void DumpWebPage()
    {
        var client = new WebClient();
        client.DownloadStringCompleted += this.OnDumpWebPageCompleted;

        Console.WriteLine("Request started at: {0}", DateTime.Now.ToString("O"));
        client.DownloadStringAsync(new Uri("http://www.mindassist.net/todo/hello"));
    }

    1 reference
    private void OnDumpWebPageCompleted(object sender, DownloadStringCompletedEventArgs e)
    {
        Console.WriteLine("Request finished at {0}", DateTime.Now.ToString("O"));
        Console.WriteLine("{0}Web-Content from mindassist: {1}", Environment.NewLine, e.Result);
    }
}
```

Wie hat man dies bisher gelöst? (3)

- Nachteile APM / EAP
 - Code-Splittung (APM & EAP)
 - Bei APM-Lambda keine Splittung → aber unleserlich
 - Event-Binding / Un-Binding (EAP)



„Async/await“ – das neue Pattern (TAP)

- Async & await als neue Schlüsselwörter
- Task / Task<T> als return Werte
- Handling innerhalb einer Methode (Async-Suffix)
- Thread-safe Dispatching (auch Exception-Stack)

Weiterführende Links

- MSDN-Dokumentation: <http://tinyurl.com/qabrn56>
- Pattern-Dokumentation: <http://tinyurl.com/od5p38u>
- Best Practice Post: <http://tinyurl.com/q489p2u>



„Async/await“ – das neue Pattern (TAP-Demo)

```
1 reference
class Program
{
    0 references
    static void Main(string[] args)
    {
        var program = new Program();
        program.DumpWebPage();

        Console.WriteLine("Waiting for response...{0}", Environment.NewLine);
        Console.ReadKey();
    }

    1 reference
    private async void DumpWebPage()
    {
        var client = new WebClient();

        Console.WriteLine("Request started at: {0}", DateTime.Now.ToString("O"));

        var result = await client.DownloadStringTaskAsync(new Uri("http://www.mindassist.net/todo/hello"));

        Console.WriteLine("Request finished at {0}", DateTime.Now.ToString("O"));
        Console.WriteLine("{0}Web-Content from mindassist: {1}", Environment.NewLine, result);
    }
}
```

Pragmatische Design Pattern: „async/await“ (1)

➤ Generell

- Eine asynchrone Methode ist für das Erzeugen des Task selbst verantwortlich
- Task.Run (ThreadPool.Queueing) besser als TaskFactory.StartNew
 - WebClient und Thread.Delay kapseln dies implizit (schlechte Beispiele 😊)
- Bei Parallelisierung → Task.WhenAll zur Task-Synchronisation
 - async all the way

Figure 9 Solutions to Common Async Problems

Problem	Solution
Create a task to execute code	Task.Run or TaskFactory.StartNew (<i>not</i> the Task constructor or Task.Start)
Create a task wrapper for an operation or event	TaskFactory.FromAsync or TaskCompletionSource<T>
Support cancellation	CancellationTokenSource and CancellationToken
Report progress	IProgress<T> and Progress<T>
Handle streams of data	TPL Dataflow or Reactive Extensions
Synchronize access to a shared resource	SemaphoreSlim
Asynchronously initialize a resource	AsyncLazy<T>
Async-ready producer/consumer structures	TPL Dataflow or AsyncCollection<T>

Pragmatische Design Pattern: „async/await“ (2)

- UI
 - Entkopplung innerhalb des View-Models
 - 2 Methoden
 - 1 Binding an ActionCommand (synchron)
 - 1 Async Executer (→ Unit-Testing)

```
private async void Authorize()
{
    await this.AuthorizeAsync();
}

public async Task AuthorizeAsync()
{
    this.IsAuthenticating = true;
    this.IsPasswordFocused = false;

    try
    {
        Logger.Write(string.Format("Trying to authorize user: '{0}'", this.UserName));

        var identity = await Task.Run(() => this.UserController.GetIdentityByCredentialsAsync(this.UserName, this.Password));

        if (!identity.IsAuthenticated)
        {
            this.Password = string.Empty;
            this.HasAuthenticationFailed = true;
            this.IsPasswordFocused = true;
        }
        else
        {
            this.DialogResultWindow = true;
            this.EventAggregator.GetEvent<IdentityChangedEvent>().Publish(identity);
        }
    }
    finally
    {
        this.IsAuthenticating = false;
    }
}
```

Pragmatische Design Pattern: „async/await“ (3)

➤ API

- Alle Schichten \leftrightarrow Nur ausgewählte Schichten (TCP, I/O, ...) ?
- Progress-Reporting / Cancellation

```
public Task MethodNameAsync(...);  
public Task MethodNameAsync(..., CancellationToken cancellationToken);  
public Task MethodNameAsync(..., IProgress<T> progress);  
public Task MethodNameAsync(...,  
    CancellationToken cancellationToken, IProgress<T> progress);
```

➤ Unit-Testing

- Deklaration: **async Task** TestMethod
- Im Test: **await** FunctionUnderTest();



Und jetzt noch...

- GitHub Link zu den Samples:
 - https://github.com/5minds/fe_async_wait
- Download-Link zur Präsentation
 - http://www.5minds.de/assets/attachments/2014_freelancer_1_async_wait_kimpenhaus_moellenbeck.ppt



Fertig! 😊

Fragen?



Jetzt oder später: buero@5Minds.de

