ASP.NET Core MVC Dependency Injection

Ingo Köster

Diplom Informatiker (FH)

Controller

- Der Controller verwendet in der Regel ein Objekt, welches den Zugriff auf die Daten eines Models erlaubt (DAO-Klasse)
- Dieses Verwaltungs- oder Repository-Objekt wird direkt in dem Controller instanziiert
- Dadurch entsteht eine sehr enge Kopplung bzw. Bindung zwischen Controller-Klasse und Repository-Klasse
- Nachteile:
  - Der Klassenname des Repositories ist hart codiert
  - Der Controller ist an dieses Objekt gebunden
  - Der Controller kann nicht ohne dieses Objekt getestet werden

Beispiel für gebundenen Controller

› Controller aus der Filmbewertungsaufgabe

```
public ActionResult Index()
{
 // DAO-Objekt
 FilmVerwaltung verwaltung = new FilmVerwaltung();

 verwaltung.LadeFilmListe();


 return View(verwaltung.AlleFilme);
}
```

Wünschenswert …

- Es ist sinnvoll das DAO-Objekt lose zu koppeln, damit
  - Eine andere Implementierung leicht eingebunden werden kann
  - In der Entwicklung ein „Dummy“ oder „Mock-up“ genutzt werden kann
  - Die Testbarkeit des Projekts sich verbessert

- Lösung:
  - Entkopplung der konkreten Bindung der Klassen durch Interfaces
  - Übergabe/Zuweisung der benötigten Objekte

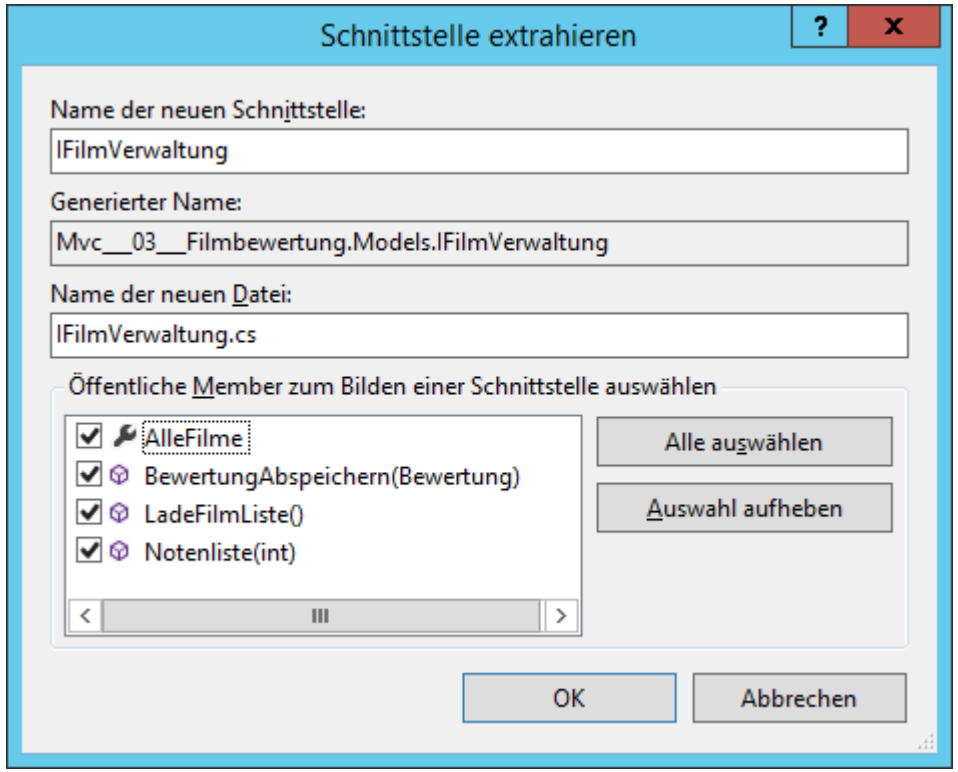
- Es gibt dafür ein Standard-Muster
  - „Dependency Injection“ oder auch „Inversion of Control“

Dependency Injection

- Ermöglicht es z.B. zwischen verschiedenen Repository-Klassen zu wechseln
  - Verschiedene Quellen: Datenbanken, Dateien, etc.
- Die Typen müssen dazu kompatibel sein
- Wird erreicht, indem eine Schnittstelle erstellt wird und diese von den jeweiligen DAO-Klassen implementiert werden

Schnittstelle extrahieren

› Mittels der Klasse Filmverwaltung eine Schnittstelle automatisch erstellen lassen



Dependency Injection

- Abhängigkeiten werden über den Konstruktor (oder Properties) zugewiesen (injizieren)
  - Konstruktor-Injektion
  - Eigenschaften-Injektion

- ASP.NET Core MVC enthält bereits Unterstützung für Konstruktor-Injektion der Controller

Veränderung des Controllers

- Der Controller wird um einen Konstruktor erweitert
- Die ActionMethoden verwenden das Objekt vom Typ der Schnittstelle für den Zugriff auf die Daten

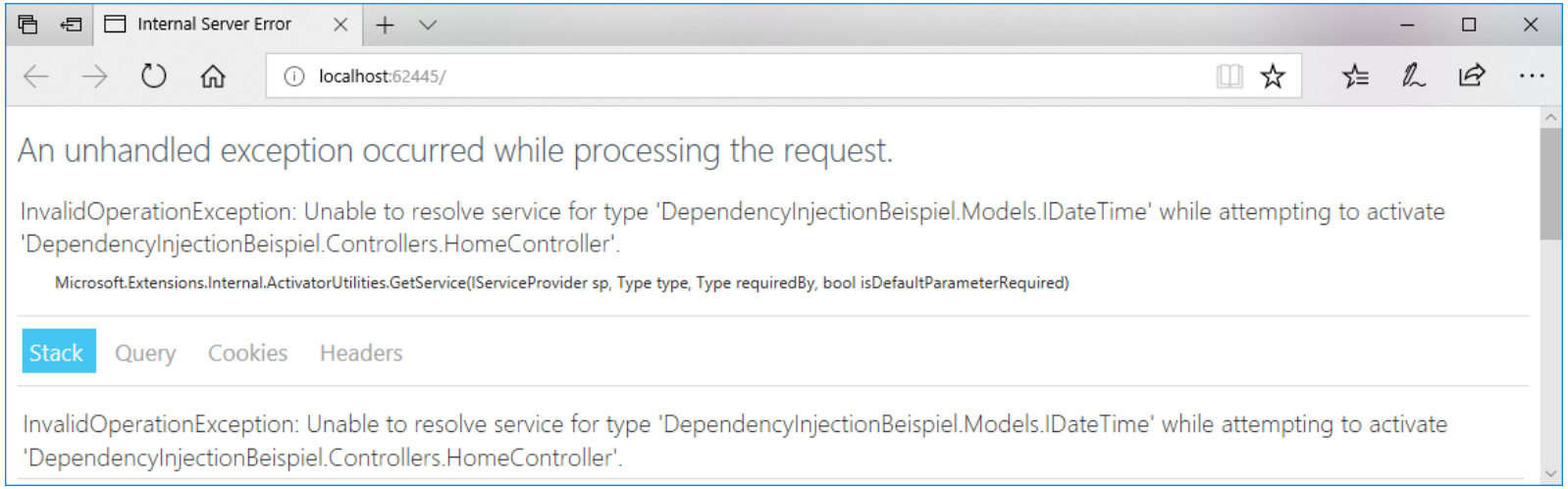
```
private IFilmVerwaltung verwaltung;
public HomeController(IFilmVerwaltung verwaltung)
{
  this.verwaltung = verwaltung;
}
```

Problem

› Der Controller kann nicht mehr von ASP.NET instanziiert werden

› Es wird ein Übergabeparameter vom Typ `IFilmVerwaltung` benötigt

› Ein konkretes Objekt wird benötigt, dieses wird als Dienst bezeichnet



Lösung des Problems

› Dieser Fehler tritt auf, wenn in der Program.cs kein Dienst konfiguriert wurde

› Damit für die verwendete Schnittstelle im Controller eine passende Instanz erzeugt werden kann, muss die folgende Zeile hinzugefügt werden

```
…
var builder = WebApplication.CreateBuilder(args);

builder.Services.AddTransient<IFilmVerwaltung,
  FilmVerwaltungObjekte>();
…
```

Lebensdauer

- Bei jedem http-Request wird ein neuer Controller instanziiert
- D.h. auch der registrierte Dienst wird bei jeden Request neu instanziiert
  - Häufig ist dieses Verhalten sinnvoll
- Für jeden der registrierten Dienste muss eine geeignete Lebensdauer ausgewählt werden
  - Für jeden Request ein neues Dienst-Objekt
  - Eine Instanz für die Dauer eines Requests (z.B. durch Umleitungen)
  - Eine Instanz für die gesamte Lebensdauer der Anwendung

Lebensdauer

- Transient (vorübergehend)
  - Vorübergehende Lebensdauer, werden bei jeder Anforderung neu erstellt
  - Für einfache, zustandslose Dienste geeignet
- Scoped (bereichsbezogen)
  - Dienste mit bereichsbezogener Lebensdauer werden einmal pro Anforderung erstellt
  - Wenn eine Instanz eines Objektes über eine bestimmte Webanforderung hindurch verwendet werden
    - Z.B. bei Weiterleitungen mittels `RedirectToAction`
- Singleton
  - Ein Objekt für die gesamte Lebensdauer der Anwendung

Singleton

- Entwurfsmuster
  - Sorgt dafür, dass von einer Klasse nur eine Instanz erzeugt werden kann
- Dienste mit Singleton-Lebensdauer werden beim ersten Request erzeugt
- In jedem nachfolgenden Request wird anschließend dieselbe Instanz verwendet
- Dafür ist keine Implementierung des Singleton-Musters in der Dienst-Klasse notwendig
  - Die Lebensdauer wird vom MVC Framework verwaltet

Beispiele

- Transient
  - `builder.Services.AddTransient<IFilmVerwaltung, FilmVerwaltungObjekte>();`
- Scoped
  - `builder.Services.AddScoped<IFilmVerwaltung, FilmVerwaltungObjekte>();`
- Singleton
  - `builder.Services.AddSingleton<IFilmVerwaltung, FilmVerwaltungObjekte>();`

Action Injection

- In einigen Fällen wird für lediglich eine Aktion innerhalb des Controllers ein Dienst benötigt
  - Bei Konstruktor-Injektion wird der Dienst für jede ActionMethode instanziiert

- In diesem Fall kann es sinnvoll sein, den Dienst als Parameter in die Aktionsmethode einzufügen

- Der Dienst wird als Parameter mit dem Attribut [FromServices] übergeben
  - `public IActionResult Index([FromServices] IDateTime dateTime) {…}`

Dependency Injection für konkrete Typen

- Dependency Injection kann auch für konkrete Typen verwendet werden
  - `builder.Services.AddScoped<FilmVerwaltungObjekte>();`

- Vorteile
  - Verwaltung der Lebensdauer
  - Erlaubt es in der gesamten Anwendung auf das Objekt zuzugreifen
  - Dem konkreten Typ können selbst wieder per Dependency Injection Dienste über den Konstruktor zugewiesen werden

Dependency Injection