



Goldschmiede

anderScore
trust in competence

Webservices und Dependency Injection – einmal ohne Framework!

24.06.2022,

Tobias Lohmüller, Daniel Krämer, Jan Lühr

© 2022 anderScore GmbH

1. Vorstellung

3

2. Was macht ein Webservice Framework?

5

3. Was spricht gegen ein Webservice Framework?

6

4. Funktion: Web & Dependency Injection

7

5. Diskussion


11

Tobias Lohmüller (B.Sc. Informatik)

- Full Stack Entwickler / Software Engineer
- Seit 2017 bei anderScore
- Schwerpunkte
 - Web Engineering
 - Online Services



Individuelle Anwendungsentwicklung - Java enterprise, web, mobile

- seit 2005 ♦ in Köln ♦ für alle Branchen ♦ **Goldschmiede**  anderScore
- nach Aufwand & im Festpreis
- ✓ Digitalisierung/ Prozesse/ Integration
- ✓ Migration
- ✓ Neuentwicklung
- ✓ Notfall/ kritische Situation
- ➔ pragmatisch, zielgerichtet, zuverlässig

Kompletter SW Life Cycle

- Projektmanagement/ agile Methodik
- Anforderungsanalyse
- Architektur & SW-Design
- Implementierung & Testautomation
- Studien & Seminare



... und für Sie? Sprechen Sie uns an!

2. Was macht ein Webservice Framework?

1. Build und Struktur

- Gradle/Maven + Konventionen

2. Web

- Routing (URL → Klasse/Methode), HTTP
- Serialisierung & Deserialisierung
- Aufbauende Eigenschaften (Authentifizierung, Autorisierung, Statistiken)

3. Dependency Injection

- Konfiguration anwenden (z.B. Datenbankzugangsdaten)
- Services konsistent instanziiieren
- Reduktion von Boilerplate Code
- Vereinfachtes Testen durch Mocking (z.B. Mockito)



3. Was spricht gegen ein Framework?

- Limitierungen (passt nicht zu 100% zum Anwendungsfall)
- Einarbeitungsaufwand (Dokumentationen, Schulung, ...)
- Security (viele Dependencies, hohe Komplexität → große Angriffsfläche)
- Ressourcenverbrauch (RAM, Startzeit, CPU) z.B. Spring @ComponentScan über viele Klassen

→ Motivation:

- Verständnis für die Paradigmen und Patterns eines Frameworks
- Begründete Technologieauswahl (Framework, Library)

1. Klassisch Java HttpServlet

- Mapping URL → doPost / doGet, ...
- Java API gibt Zugriff auf HTTP Daten
- Abstraktion: Strings und I/O Streams
- „Altes Java“ keine Frameworks / externen Bibliotheken erforderlich



2. Webservice Framework

- High-Level API (JAX-RS: Annotationen)
- Häufig externe Bibliothek implementiert API (z.B. Spring-Servlet)
- Teilweise alternative Grundlage vgl. mit Servlets (Vert.x)
- Abstraktion: Strukturierte Objekte & Serialisierung (JSON / XML)

Fazit: Abstraktion hilft bei strukturierten Daten, z.T. aber aufwendige Konfiguration

1. Herausforderung: Objekte konsistent erstellen

- Scope: Singleton, Request, Session
- Konfiguration richtig anwenden
- Abhängigkeiten bei der Erstellung berücksichtigen



2. Klassisch: Factory-Pattern

- Erstellung aller Objekte mit *new* in einer Factory-Klasse (ggf. mehrere)
- Factory liest Konfigurationsdateien ein
- Reihenfolge der new-Operationen berücksichtigt Abhängigkeiten

3. Webservice Framework: Dependency Injection

- XML, Annotationen, Java Code ersetzen Factory
- Framework instanziiert in einem Container (quasi Factory)
- Framework übernimmt Kontrollfluss (Inversion of Control)
- Häufig: ContainerServlet oder ServletFilter



Fazit:

- Factory-Klassen umfangreich und mühselig zu schreiben
- DI Container häufig mit hohem Funktionsumfang und vielen Optionen
→ Fehleranfällig, schwer zu debuggen, aufwendig zu lernen

Danke!

Vielen Dank fürs Zuhören!

- Welche Frameworks benutzt ihr?
- Wie viele Zeilen Code hatte eure längste Application Factory?
- Gab es eine Situation, wo ihr auf ein Framework verzichtet?
- Wann würdet ihr auf ein Framework zugreifen, wann nicht?



- <https://www.redbubble.com/de/i/t-shirt/Abhängigkeitsspritze-von-cdemi/50070896.UIS2>
- <https://betterprogramming.pub/building-a-simple-spring-like-dependency-injection-framework-in-java-5d91254d2dbf>