Fachvortrag Web

spring

by Pivotal™

Team Imperium: Mohamad Kobeissi, Sandro Hoffmann, Gavin Auerswald , Sarah Schulte

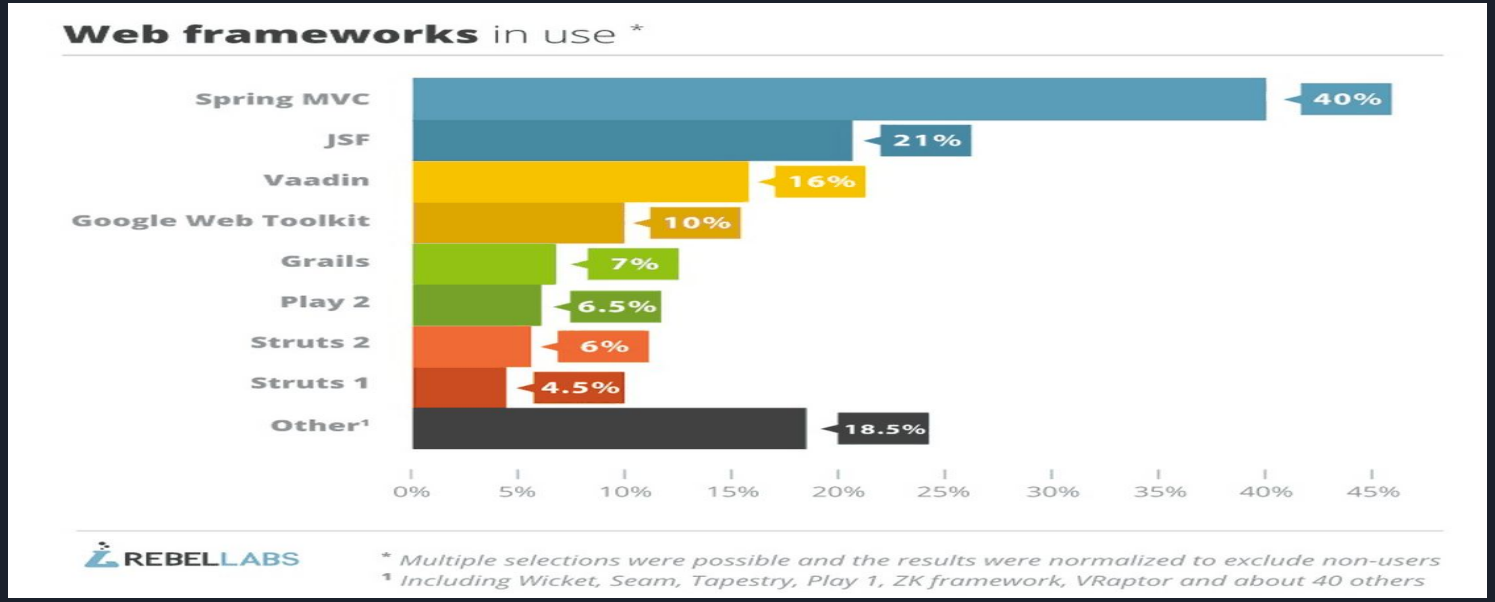
Fachvortrag Web: Spring

1. Einleitung
2. Dependency Injection
3. Aspektorientierung
4. Code-Beispiel

Einleitung

- Erfinder: Rod Johnson
- quelloffenes Framework
- Ziel
- Eigenschaften
- Erweiterungen
- MVC
- usage Statistiken

Statistiken



Dependency injection

- ist ein Entwurfsmuster der objektorientierten Programmierung
- Abhängigkeiten werden injiziert
- Code unabhängiger von Umgebung
- Spring Injizierung in xml Datei
- mehrer Arten: Constructor, Setter, Annotationen
- jede injizierte bean muss Klasse in Java besitzen

Constructor injection

- Konstruktor arbeitet einfachen Datentypen
- wenn mehrere Konstruktoren willkürliche Auswahl
- in XML-Dateien Datentyp definieren (bsp. type = “java.lang.string”)
- beschreibung mittels index attribut (bsp. index = “0”)
- einen konstruktor in java code anbieten
- Verwendung bei nicht optionalen Objekten

Code Beispiel Constructor Injection

- in der XML-Datei

```
<bean id=”hello" class="helloWorld.Text">
        <constructor-arg type=”java.lang.String” value=”hallo Welt>
</bean>
```

Setter injection

- Methoden stellen Abhängigkeiten zur verfügung
- wird für optionale Abhängigkeiten genutzt
- Zuordnung der Werte eindeutiger als bei Constructor Injection
- erneute Injektionen besser realisierbar
- Attribute können vergessen werden

Code Beispiel Setter Injection

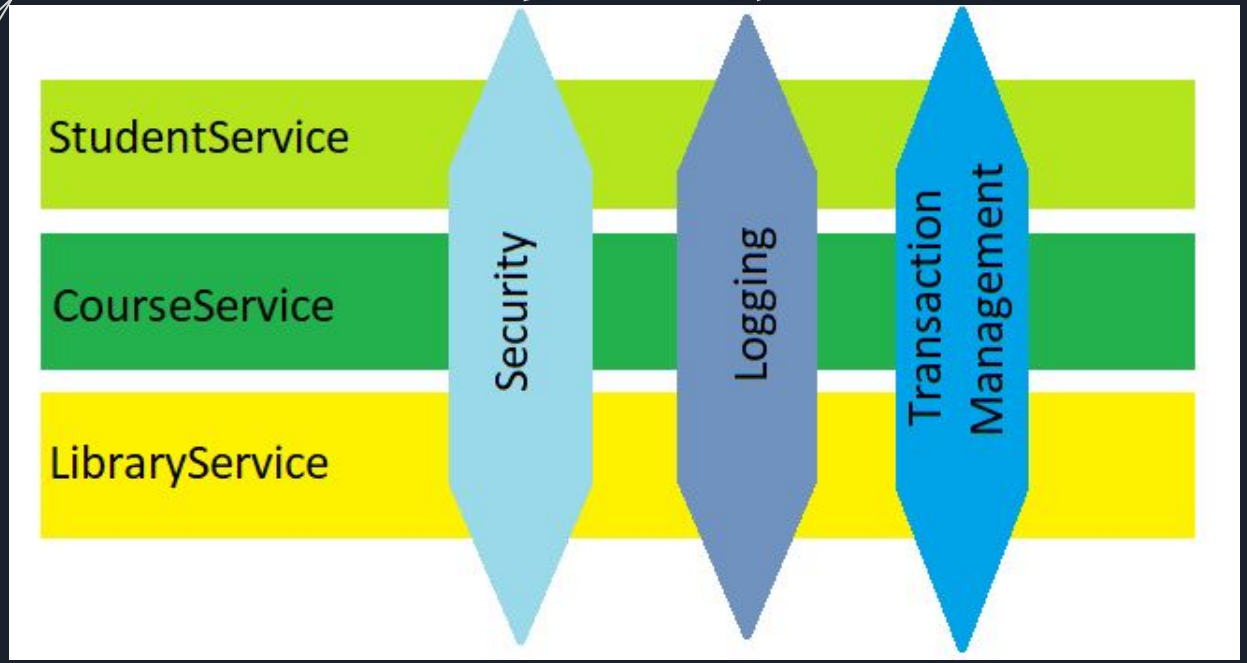
- in der XML-Datei

```
<bean id="hello2" class="helloWorld.Text">
        <property text="title" value="hallo Welt" />
</bean>
```

Aspekt-Orientierung

core concerns

cross cutting concerns = aspects



Grafik nach Walls (2011): Spring in Action, S. 86

```
public int enrollStudent (args)
{
        Student s = new Student (args);
        log.write (“Enrolling “ + s);
        dbmanager.prepareTransaction();
        matrikel = db.insert(s)
        dbmanager.endTransaction();
        log.write ("enrolled: " + matrikel);
        return matrikel;
}
```

```
public int enrollStudent (args){
        Student s = new Student (args);
        matrikel = db.insert(s)
        return matrikel;
}

public class Logging {
        public void write {...}
}

aspect_logging {
    before (db.insert(s)):{
        log.write (“Registering “ + s);
    }
    after (db.insert(s)): {
        log.write ("Registered: " + matrikel);
    }
}
```

Hinzufügen von Funktionalität eines Aspekts (advice) auf 5 Arten:

- before: vor Methodenaufruf
- after: nach Methodenaufruf
- around: schließt die Methode ein
- after-returning: nach erfolgreichem Abschluss der Methode
- after-throwing: nach Exception

→ in Spring immer auf Methodenlevel, aber auch möglich für Konstruktoren und Felder (z.B. mit AspectJ)

→ potentielle Ansatzpunkte für advice heißen join points

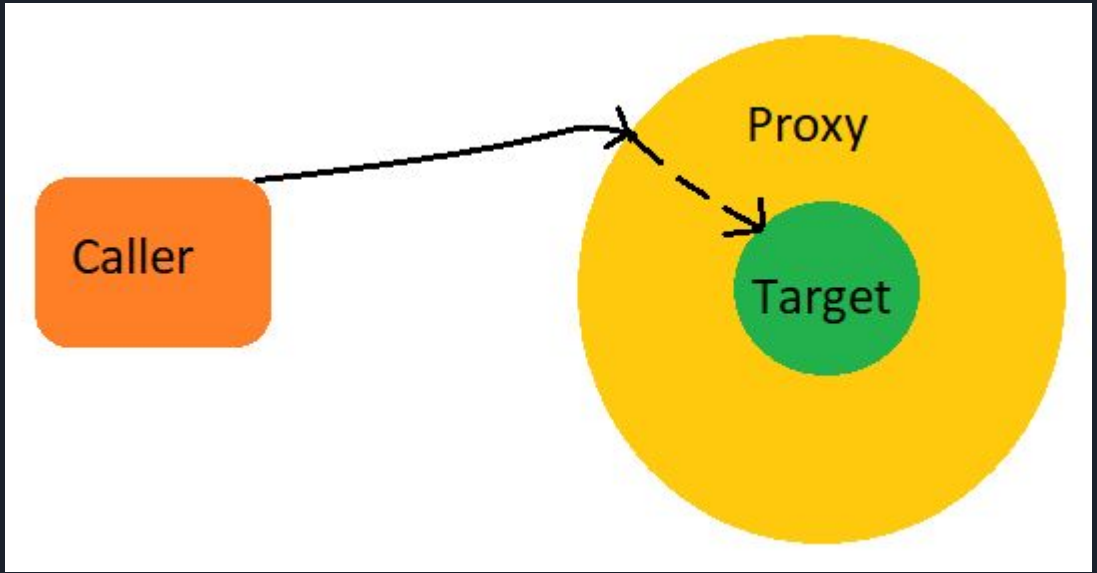
→ Definitionen *tatsächlicher* Ansatzpunkte (bestimmte Klassen-/Methodennamen) heißen pointcuts

Beispiel: Einbinden des Logging via XML

```
<aop:config>
    <aop:aspect ref=”logging”>
        <aop:before pointcut =
            “execution(* com.HTW.studentService.Student.enroll(..))”
            method = “write” />
        <...>
    </aop:aspect>
</aop:config>
```

Weaving

= Anwenden von Aspekten auf ein konkretes Objekt → Proxy-Objekt



→ in Spring:
nur zur Laufzeit

→ mit AspectJ
auch zur
Kompilierzeit oder
beim Klassenladen

Grafik nach Walls (2011): Spring in Action, S. 90