

Das Spring Framework für JavaEE

<http://www.springframework.org>



Autor: Michael Schmidt

Was ist das Spring Framework?

- Spring ist ein Lightweight **Application** Framework
- Betrachtet man Struts, WebWork und andere als **Web** frameworks, so umfasst Spring alle Schichten einer Applikation
- Spring ist KEIN Application Server, aber kann gut mit solchen (oder anderen Java Umgebungen) integriert werden. Spring kann in vielen Fällen Dienste elegant ersetzen, die traditionell von J2EE Application Servern geboten werden
 - Service Lookup
 - Persistency / Transaction Handling
- Spring liefert Basisinfrastruktur für die Vernetzung der Komponenten, so dass die Entwicklung sich ganz auf *diese* konzentrieren kann!

- Begonnen 2002/2003 durch Rod Johnson und Juergen Holler
- Anfangs als Framework um Rod Johnson's Buch *Expert One-on-One J2EE Design and Development*
- Spring 1.0 Released März 2004
- 2004/2005 Spring verbreitet sich schnell und wird zum *führenden* Java/J2EE Application Framework
- Spring 2.5 released November 2007



The Spring Framework Mission Statement

Von *springframework.org*

- **We believe that:**

- J2EE should be easier to use
- It's best to program to interfaces, rather than classes. Spring reduces the complexity cost of using interfaces to zero.
- JavaBeans offer a great way of configuring applications.
- OO design is more important than any implementation technology, such as J2EE.
- Checked exceptions are overused in Java. A framework shouldn't force you to catch exceptions you're unlikely to be able to recover from.
- Testability is essential, and a framework such as Spring should help make your code easier to test.

Von *springframework.org*

We aim that:

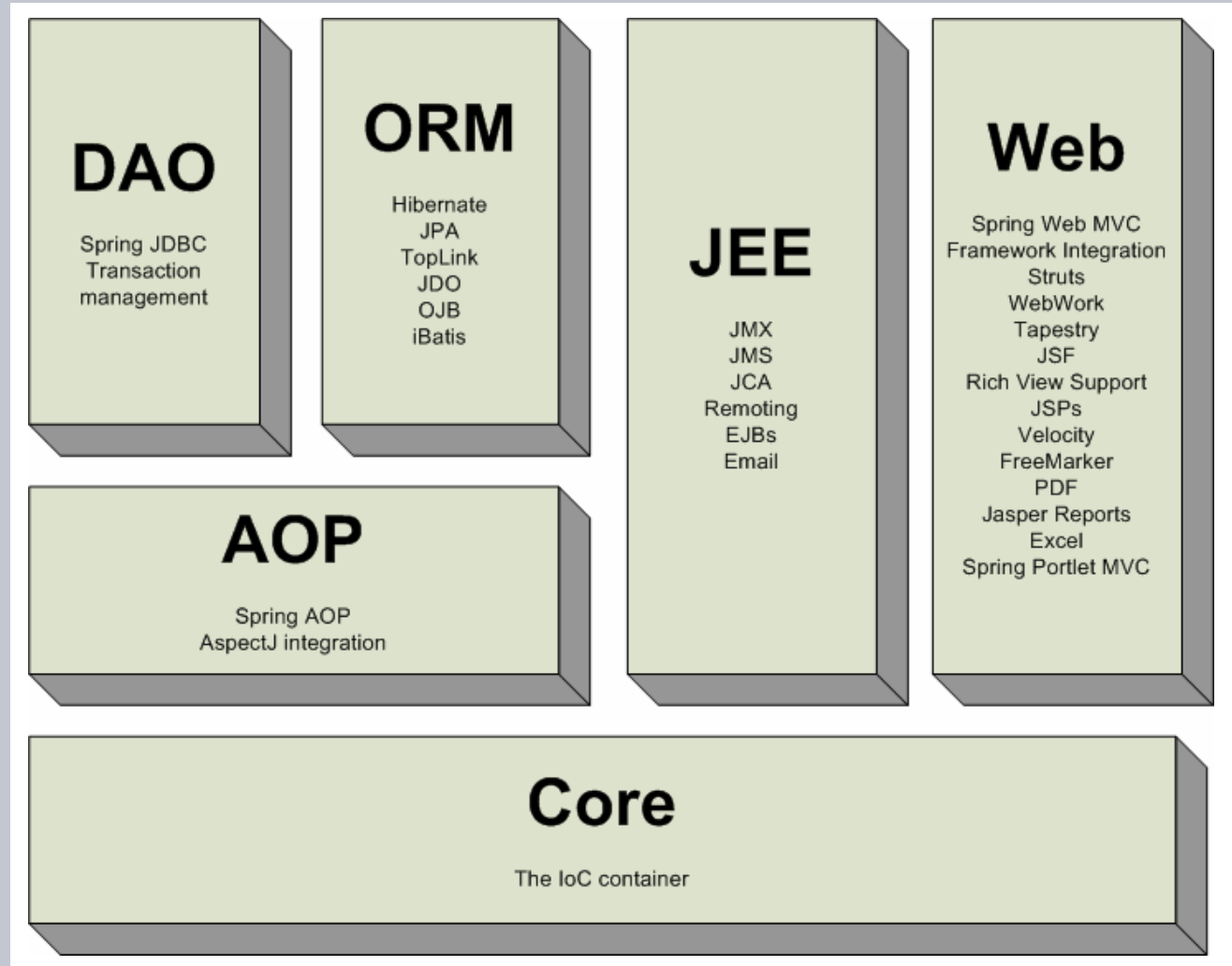
- Spring should be a pleasure to use
- Your application code should **not** depend on Spring APIs
- Spring should not compete with good existing solutions, but should foster integration. (For example, JDO and Hibernate are great O/R mapping solutions. We don't need to develop another one.)

- Invasive APIs behindern die Entwicklung.
- Negativ-Beispiele:
 - EJB erfordert die Benutzung von JNDI
 - Struts fordert die Erweiterung von `Action`
- Invasive Frameworks sind schwierig zu testen, da immer eine Runtime benutzt oder emuliert werden muss, die durch den Application Server geliefert wird.
- Stärke von Spring: Entwickler müssen keine Spring APIs importieren oder erweitern.

Was ist Spring konkret?

- Im Kern liefert Spring:
 - einen **I**nversion of **C**ontrol Container
 - auch bekannt als **D**ependency **I**njection (Martin Fowler)
 - ein AOP Framework
 - Spring bietet ein Proxy-basiertes AOP framework
 - alternativ kann es mit AspectJ oder AspectWerkz integriert werden
 - einen Service Abstraction Layer
 - konsistente Integration mit vielen Standard und 3rd Party APIs
- Dies befördert die Entwicklung von mächtigen und skalierbaren Applications auf Basis von POJOs.

Spring Overview



from springframework.org

Inversion of Control

- Bei dem Design Pattern IoC sind Java-Klassen nicht per Java-Code fest verknüpft, sondern werden von einem Framework aufgerufen.
- The *Hollywood Principle*: Don't call me, I'll call you

Dependency Injection (DI)

- DI ist ein von Martin Fowler definiertes zu IoC nahezu identisches Design Pattern. DI hat weniger explizite Abhängigkeiten zum Container-API als IoC. Konfigurationsparameter und zu verwendende andere Beans werden vom Framework zur Laufzeit bei der Instanziierung von Java-Objekten **injiziert**.
- Man unterscheidet zwischen *Setter Injection* und *Constructor Injection*.

Vorteile

- eliminiert direkte Abhängigkeiten von Klassen
- eliminiert Lookup Code innerhalb der Applikation
- ermöglicht Pluggability und Hot Swapping
- unterstützt gutes OO Design
- erleichtert die Wiederverwendbarkeit von Code
- verbessert die Testbarkeit dramatisch

- **BeanFactories** sind der Kern von Spring
- **ApplicationContext** ist eine BeanFactory, aber enthält zusätzliche Framework Features
 - i18n messages
 - event notifications
- Eine **BeanFactory** wird üblicherweise durch eine XML-Datei mit dem Root-Element `<beans>` konfiguriert.
- Das Root-Element enthält `<bean>`-Elemente
 - `id` (oder `name`) attribute to identify the bean
 - `class` attribute to specify the fully qualified class

- Beispiel (Setter Injection)

```
<beans>  
  
  <bean id="dataSource" class="...ClientDataSource">  
    <property name="server" value="localhost" />  
    <property name="port" value="1234" />  
  </bean>  
  
  <bean id="supplierDAO" class="...SupplierDAO">  
    <property name="dataSource" ref="dataSource" />  
  </bean>  
  
</beans>
```

- Standard sind Singletons
- Alternative Prototypen (`scope="prototype"`)

Alternative Inline Konfiguration von Referenzen

```
<beans>

  <bean id="supplierDAO" class="...SupplierDAO">
    <property name="dataSource">
      <bean class="...ClientDataSource">
        <property name="server" value="localhost" />
        <property name="port" value="1234" />
      </bean>
    </property>
  </bean>

</beans>
```

Alternative Konfiguration mittels Konstruktor-Argumenten (Constructor Injection)

```
<bean id="dataSource" class="...ClientDataSource">  
  <constructor-arg value="localhost" />  
  <constructor-arg value="1234" />  
</bean>
```

```
<bean id="supplierDAO" class="...SupplierDAO">  
  <constructor-arg ref="dataSource" />  
</bean>
```

```
<bean id="dataSource" class="...ClientDataSource">  
  <constructor-arg type="String" value="localhost" />  
  <constructor-arg type="int" value="1234" />  
</bean>
```

```
<bean id="dataSource" class="...ClientDataSource">  
  <constructor-arg index="0" value="localhost" />  
  <constructor-arg index="1" value="1234" />  
</bean>
```

Collections (List, Set, Map, Properties)

```
<bean id="myList" class="MyList">
  <property name="list">
    <list>
      <value>First List Entry</value>
      <value>Second List Entry</value>
    </list>
  </property>
</bean>

<bean id="myMap" class="MyMap">
  <constructor-arg>
    <map>
      <entry key="1" value="Hello"/>
      <entry key="2" value="World"/>
    </map>
  </constructor-arg>
</bean>
```

init & destroy

```
<bean id="supplierDAO"  
      class="...SupplierDAO"  
      init-method="init"  
      destroy-method="cleanup"/>
```

```
public class SupplierDAO {  
    public void init() {...}  
    public void destroy() {...}  
}
```

Factory Methods

```
<!-- static factory methode -->  
<bean id="exampleBean"  
      class="ExampleBean2"  
      factory-method="createInstance"/>
```

```
<!-- instance factory methode -->  
<bean id="myFactoryBean" class="MyFactory"/>  
<bean id="exampleBean"  
      factory-bean="myFactoryBean"  
      factory-method="createInstance"/>
```

Autowiring

- no
- byName
- byType
- constructor
- autodetect

```
<beans>

  <bean id="dataSource" class="...ClientDataSource">
    <property name="server" value="localhost" />
    <property name="port" value="1527" />
  </bean>

  <bean id="supplierDAO"
    class="...SupplierDAO"
    autowire="byName" />
  <!-- property dataSource will be set by autowiring -->
</beans>
```

Instanziierung

```
public void main(Object[] args)
{
    Ressource resource =
        new FileSystemRessource("beans.xml");
    BeanFactory factory = new XmlBeanFactory(resource);

    Object o = factory.getBean("myBean");
}
```

```
public void main(Object[] args)
{
    ApplicationContext context =
        new FileSystemXmlApplicationContext("beans.xml");

    Object o = context.getBean("myBean");
}
```

- Verwaltung und Implementierung allgemeiner querschnittlicher Aspekte (Cross Cutting Concerns) an zentraler Stelle, ohne dass in den vielen betroffenen Java-Klassen Code implementiert werden muss.
 - Tracing
 - Logging
 - Transaktionssteuerung
 - Sicherheit

- Ein **Advice** ist eine per *Interception* eingeschleuste Aktivität.
 - AroundAdvice: als MethodInterceptor rund um Methodenaufrufe mit beliebiger Funktionalität
 - BeforeAdvice: vor den Methodenaufrufen
 - AfterReturningAdvices: nach den Methodenaufrufen (kann aber Rückgabewerte nicht modifizieren)
 - ThrowsAdvices - fängt Exceptions

- Ein **PointCut** definiert, für welche Beans oder Methoden ein Advice eingesetzt werden soll (z.B. mit *JdkRegexpMethodPointcut*).
- Ein **Advisor** kombiniert die Advices und Pointcuts (z.B. *DefaultPointcutAdvisor*)

- *Aspekte* werden in eigenen Klassen definiert und per *Interception* in POJOs integriert.
- Bei Verwendung von Interfaces wird AOP über *Dynamic Proxies* implementiert.
- Ohne Interfaces werden mit Hilfe von CGLIB Subklassen generiert, deren Methoden die Advices und die Originalmethoden aufrufen (vorausgesetzt die Originalklasse ist nicht final).

- Pre-packaged AOP services
 - Declarative Transaction Management
 - Security
 - Logging
- Potentielle eigene AOP services
 - Auditing
 - Caching
 - Custom security

```
<beans>

  <bean id="idMyBean" class="sampleaop.MyBean" />
  <bean id="idMyDAOBean" class="sampleaop.MyDAOBean" />

  <bean id="idMyAdvice" class="...MyMethodInterceptor" />

  <bean id="idMyPointcut"
    class="org.springframework.aop.support.JdkRegexpMethodPointcut">
    <property name="pattern" value=".*aop.MyBe.*" />
  </bean>

  <bean class="org.springframework.aop.support.DefaultPointcutAdvisor">
    <property name="advice" ref="idMyAdvice" />
    <property name="pointcut" ref="idMyPointcut" />
  </bean>

  <bean class="org.springframework...autoproxy.DefaultAdvisorAutoProxyCreator"
/>

</beans>
```

Spring bietet Abstraktion für diverse Services

- Transaction Management
 - JTA, JDBC, others
- Data Access
 - JDBC, Hibernate, JDO, TopLink, iBatis
- Email, Messaging
- Remoting
 - EJB, Web Services, RMI, Hessian/Burlap

Nutzen

- keine “implicit contracts” mit JNDI
- Abgrenzung von genutzten APIs
- bessere Wiederverwendbarkeit
- Realisierung komplett über Interfaces
- vereinfachtes Testen

- Generische DataAccessException-Hierarchie, die von JDBC, Hibernate, JDO, etc abstrahiert
 - unchecked Exceptions, da für die meisten eine Fehlergehandlung unmöglich ist
 - Vermeidung sinnloser try/catch-Blöcke
- Vereinfachung von DAO Implementierungen durch Support-Klassen
 - z.B. JdbcTemplate, HibernateTemplate, JdoTemplate
 - viele Operationen werden zu Einzeilern

```
public class ExampleJdbcDao extends JdbcDaoSupport
{
    public void clearDatabase() throws DataAccessException
    {
        getJdbcTemplate().update("DELETE FROM imagedb");
    }

    public void deleteImage(int imageId) throws DataAccessException
    {
        getJdbcTemplate().update("DELETE FROM imagedb WHERE id=?",
            new Object[] {new Integer(imageId)});
    }

    public int getNrOfImages() throws DataAccessException
    {
        return getJdbcTemplate().
            queryForInt("SELECT COUNT(*) FROM imagedb");
    }
}
```

- Spring bietet Möglichkeiten zur einfachen Intergration mit Struts, WebWork, JSF, Tapestry, Velocity und anderen Web Frameworks
- Außerdem bietet Spring ein integriertes Web Framework - **Spring MVC**

- Das Spring MVC Framework bietet eine einfache Interface-basierte Infrastruktur für die Realisierung von Web-MVC-Anwendungen
- Spring MVC Komponenten sind *Standard* Spring Beans
 - Andere Spring Beans können in die Spring MVC Komponenten *injiziert* werden
 - Spring MVC Komponenten können einfach getestet werden (ohne Application Server)

Controller

Implementation von

```
org.springframework.web.servlet.mvc.Controller  
ModelAndView handleRequest(request, response)
```

Das Basis Controller Interface - vergleichbar mit einer Struts Action.

View

Implementation von

```
org.springframework.web.servlet.mvc.View  
void render(model, request, response)
```

MVC view für die Web Interaktion. Implementierungen sind verantwortlich für das Rendern des Seiteninhalts und die Aufbereitung der Daten des Modells.

Model

Das Modell ist üblicherweise eine `java.util.Map` die dem View zur Verfügung gestellt wird.

Zugriff in JSP: `<jsp:useBean/>` mit **id** als Schlüssel innerhalb der Map

Spring kann auf zwei Wegen mit Web Frameworks integriert werden:

Lookup von Spring Beans innerhalb von Controller/Action-Instanzen mittels:

```
WebApplicationContextUtils
```

```
.getWebApplicationContext (servletContext)  
    .getBean ("beanName")
```

Konfiguration von Controller/Action-Instanzen des Web Framework in einer Spring BeanFactory und Einsatz von Spring Proxies in der eigentlichen Web Framework Konfiguration wenn verfügbar ist diese Methode zu bevorzugen
bessere Modularisierung/Testbarkeit durch Dependency Injection

- In einem aktuellen Portal-Projekt, das mittels Struts 1.3.8 realisiert wird, sind diverse externe Dienste (Java-Komponenten) in das System zu integrieren. Diese Dienste sind während der Entwicklung des Portals nicht verfügbar, sondern werden durch Dritte entwickelt und im Produktiv-System des Kunden zum Einsatz gebracht.
- Entsprechend wird Spring in dem Projekt unter anderem eingesetzt, um per Konfiguration Dummy-Dienste anstelle der echten Komponenten in das System zu integrieren.
- Am Beispiel einer Payment-Komponente wird die Realisierung exemplarisch gezeigt.

- **Design Payment-Funktionalität**
- Das vereinbarte Interface

```
public interface PaymentService {  
    ...  
    public abstract OrderResponseElement  
        authorizeAndCapture(...) throws PaymentException;  
    ...  
}
```

- **Design Payment-Funktionalität**
- Die Verwendung in der Struts-Aktion

```
public class BuySomethingAction
    extends AuthorizedDispatchAction
{
    /** The payment interface service. */
    private PaymentService paymentService;
    ...

    ...
    try {
        paymentResponse =
            paymentService.authorizeAndCapture(...);
    } catch (PaymentException e) {...}
    ...
}
```

- **Spring Konfiguration (applicationContext-struts.xml)**
- konfiguriert den Service Dummy und integriert ihn in die Aktion

```
<beans>
  ...
  <bean name="/BuySomething"
        class="...BuySomethingAction">
    <property name="paymentService" ref="paymentService" />
    ...
  </bean>

  <bean id="paymentService"
        class="...PaymentServiceMockImpl">
    <description>
      The Payment Service Dummy Service
    </description>
  </bean>
  ...
</beans>
```

- **Struts-Konfiguration (struts-config.xml)**

```
<struts-config>

<plug-in className="org.springframework.web.struts.ContextLoaderPlugIn">
  <set-property property="contextConfigLocation"
    value="...,classpath:applicationContext-struts.xml" />
</plug-in>

...
  <action
    path="/BuySomething"
    type="org.springframework.web.struts.DelegatingActionProxy"
    name="buysomethingform"
    parameter="method"
    ...>
    ...
    <forward name="BuySomethingPayment"
      path="/buy/BuySomethingPayment.jsp" />
    ...
  </action>
...
</struts-config>
```

- **Alternative Spring-Konfiguration**
- Produktiver Dienst

```
<bean id="paymentService"  
      class="...PaymentServiceImpl">  
  <description>The REAL Payment Service</description>  
  <property name="urlString"  
            value="{paymentInstance.urlString}" />  
  <property name="merchantID"  
            value="{paymentInstance.merchantID}" />  
  <property name="terminalID"  
            value="{paymentInstance.terminalID}" />  
  ...  
</bean>
```

Spring Framework Step-by-Step Tutorial

<http://www.springframework.org/docs/Spring-MVC-step-by-step/index.html>

Some samples about an entity with different implementation of a DAO interface plus various View technologies

<http://www.zabada.com/technology/Wiki.jsp?page=SpringRecipes>

Mini Samples for various technology aspects of Spring

<http://torsten-horn.de/techdocs/jee-spring.htm>

The Official Spring Reference Manual

<http://www.springframework.org/docs/reference/>

Introduction to Spring by Rod Johnson

[http://www.theserverside.com/articles/article.tss ->
?l=SpringFramework](http://www.theserverside.com/articles/article.tss?l=SpringFramework)

Spring in Action by Craig Walls and Ryan Breidenbach

Pro Spring by Rob Harrop and Jan Machacek

J2EE Without EJB by Rod Johnson and Juergen Holler

Expert One-on-One J2EE Design and Development by Rod Johnson

Better, Faster, Lighter Java by Bruce Tate and Justin Gehtland

- gute Umsetzung der Ziele des Mission Statements
- umfangreiche Integration mit gängigen Technologien
- hohe Stabilität
- große Akzeptanz in Entwicklerkreisen
- hohe Verfügbarkeit von Knowhow
- wegen geringem Overhead auch für kleinere Projekte (und sukzessive) einsetzbar
- gut geeignet für Integrationsprojekte