

Het Spring Framework



Alef Arendsen (alef@jteam.nl)

JTeam B.V. - Amsterdam

Terminologie

- Inversion of Control / Dependency Injection
- J2EE versus EJB
- ApplicationContext, bean, BeanFactory
- AOP, aspect, joinpoint, advice, pointcut
- JSR 175 annotations; @Session, @Resource

Spring framework visie

- Maak J2EE gemakkelijker om te gebruiken
- Adresseer end-to-end requirements, niet maar 1 tier
- Voorkom de noodzaak voor middle tier “glue”
- Aanleveren van de beste Inversion of Control container
- Aanleveren van een pure Java AOP implementatie, met een focus op het oplossing van alledaagse problemen (bijv. transaction management)
- Volledig *portable* tussen alle applicatie servers (van Servlet containers als Tomcat en Resin tot volledige app.servers als BEA WebLogic en IBM WebSphere)
 - Maar: niet alleen op een applicatie server. *Fat client* ook mogelijk

...Spring framework visie

- “**Non-invasive**” framework
 - Application code geen of minimale afhankelijkheden op Spring API
 - **Basis voor heel het design van Spring**
 - *Power to the POJO*
- Faciliteren van unit testing
 - Vergemakkelijken van Test Driven Development (TDD)
 - Mogelijk maken om business objects buiten de container te testen
- Faciliteren van gebruiken van OO design patterns en best practices
 - We zijn gewend ‘J2EE’ of ‘EJB’ applicaties te schrijven, geen OO applicaties: ***dit hoeft niet zo te zijn!***
- Een goed alternatief voor EJB voor *de meeste* applicaties
- Verhogen van productiviteit vergeleken bij ‘traditionale’ J2EE applicaties

The new wave of frameworks

- Grote veranderingen in de manier van ontwikkeling van J2EE applicaties
- Minder nadruk op EJB
- Niet alleen Spring: PicoContainer, HiveMind en andere *Inversion of Control* frameworks
 - 2004: Het jaar van de lightweight frameworks
 - EJB 3.0 (2005) programming model lijkt *erg* veel op **Spring (IoC minus veel features) + Hibernate**
 - Natuurlijk heeft EJB nog een aantal unieke mogelijkheden (distributed trans. Management, RMI remoting), echter voor een klein gedeelte van de applicaties
- De lightweight frameworks zijn *wel degelijk* anders
- **Spring is de meest volwassen, meest krachtige, populairste**

Unieke Spring features

- Declaratieve transaction management voor POJOs *zonder* EJB
- Consistente aanpak voor *data access* met gemeenschappelijk *exception* hiërarchie
 - Simplificeert werken met Hibernate, JDO, JDBC, etcetera
- Flexibel en *non-intrusive* MVC framework
- **IoC/AOP integratie**
- Integratie met een grote verscheidenheid aan populair producten
- Gestalt
 - Meer dan alleen de verschillende onderdelen
 - Slecht te vangen in een one liner, onze gebruikers vinden het echter geweldig
- Solide, robust en het werkt *nu*
- In productie in mission-critical applicaties *op dit moment*

Een gelaagd framework

- Web MVC
- AOP framework
 - Integreert met IoC
- **IoC container**
 - *Dependency Injection*
- Transaction management
- Data access
- *One stop shop* maar *wel modulair*

Web MVC

- Lijkt qua design erg op Struts
 - Een gemeenschappelijk Controller die alle requests afhandelt voor een specifiek request
- *Controllers, interceptor draaien in de IoC container (later meer)*
 - Belangrijke en onderscheidende feature
 - Spring *eats it own dog food*

Web MVC: voordelen t.o.v. Struts

- Meerdere front controller servlets mogelijk
 - Eleganter dan zelfs de Struts 1.1 aanpak
 - DispatcherServlets kunnen application context delen
- Veel flexibeler
 - Niet class maar interface-based
- Gemakkelijk te customizen
- Minder afhankelijk van JSP
 - Velocity, Excel, PDF etc
 - Gemakkelijk om zelf views te implementeren

Web MVC: voordelen t.o.v. Struts

- Schone MVC implementatie
 - Duidelijk onderscheid tussen model, view en controller
 - Model is niet afhankelijk van Servlet API
- Geen custom ActionForms meer maar hergebruik van domain model of TOs
- *Concrete inheritance is the last thing you want to burn up in Java. It was the first thing Struts burnt up* (Erik Hatcher)
- Makkelijk te unit testen doordat het interface-based is
- Integreert naadloos met middle-tier; **geen custom coding**
 - Nooit meer Service Locators, nooit meer custom Singletons

Web integratie

Maar ik wil WebWork/Tapestry/Struts/JSF/whatever

- Weet je wel zeker dat je Struts wil????
- **Spring: de klant is koning!**
 - ...tenzij de klant nooit iets anders heeft geprobeerd dan Struts
- Nu serieus: Spring dicteert niet wat je moet gebruiken
 - Gedane investeringen nooit zonder reden weggooien
 - **Refactor de gewenste onderdelen naar Spring**
- Integratie met Tapestry, WebWork en Struts is naadloos
- Onderweg: JSF en Portlet API support

Spring in de middle-tier

- Complete oplossing voor managen van business objects
 - Schrijf alles in POJOs
 - Spring regelt het *wiring* process en de lifecycle
 - Simpel en consistent XML formaat (commonest choice)
 - XML is echter niet het enige waarmee de IoC container werkt!
- Eigen code heeft weinig afhankelijkheden op de container—vaak geen afhankelijkheden zelfs
 - Spring Pet Store heeft *geen enkele* afhankelijkheid op Spring
 - Geen magische annotaties voor IoC, trans. management
- Gemakkelijk unit testen, TDD werkt!

Middle-tier: Dependency Injection

- **De meest complete IoC container**
 - Setter Dependency Injection
 - Configuratie via JavaBean properties
 - Constructor Dependency Injection
 - Configuratie via constructor argumenten
 - Zoals PicoContainer
 - Dependency lookup
 - Avalon/EJB callbacks
 - Ik prefereer Setter injection, maar: de klant is koning, jij kiest
 - Binnen framework ontwikkeling is geen plaats voor ideologen
 - Een goede IoC container moet omwille van legacy integratie alle vormen van Dependency Injection ondersteunen

Middle-tier: Dependency Injection

```
public class ServiceImpl implements Service {
    private int timeout;
    private AccountDao accountDao;

    public void setTimeout(int timeout) {
        this.timeout = timeout;
    }

    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    // Business methoden van Service interface (createAccount, suspendAccount
    . . .

    <bean id="service" class="com.mycompany.service.ServiceImpl">
        <property name="timeout"><value>30</timeout></property>
        <property name="accountDao"><ref local="accountDao"></ref></property>
    </bean>
```

Middle-tier: Dependency Injection

- Simpele of object properties
 - Configuratie (timeout)
 - Afhankelijkheden op collaborators (accountDao)
- Configuratie properties zijn ook belangrijk
- Veel classes hoeven niet veranderd te worden
- Dependency checking
- “Autowiring”
- Eenvoudig om (zonder Spring) classes te testen *buiten de container*
- Hergebruik van application classes *buiten de container*
- Hot swapping, instance pooling (met AOP)

Spring in de middle-tier

- Geavanceerde IoC features
 - FactoryBean introduceert abstractie, decoupling
 - AOP proxies
 - EJB clients *zonder code!*
 - JNDI factory bean
 - Custom factory beans
 - List, maps, sets, met arbitraire nesting
 - Standaard JavaBeans PropertyEditor support
 - Registreer eigen PropertyEditors
 - “Post processors”

Waarom AOP?

- **AOP complementeert IoC om samen een compleet non-invasive framework te creëren**
- Scheiden van *crosscutting concerns* in application code
 - Aspecten van een systeem die door het hele verticale model heen *snijden*
 - AOP biedt een significant andere manier van denken over applicaties en objecten dan OOP
- Interceptie van EJBs is conceptueel gezien gelijk, maar niet uitbreidbaar en te afhankelijk van componenten
- Spring heeft out-of-the-box aspects
 - Declarative transaction management voor elke POJO
 - Pooling
 - Resource acquisition/release

AOP definities

- **Aspect:** modularisatie van een crosscutting concern
- **Join point:** *point of execution* in een programma
 - setAge()
- **Advice:** Actie te ondernemen op een join point
 - Pas declaratieve transactionmanagement toe op rond dit join point
 - Invalideer cache
- **Pointcut:** Verzameling join points op welke een advice moet worden toegepast
 - Alle setter methodes in package com.mycompany.mypackage
 - Alle methodes op dit target object (instantie van class) (default)
- **Introduction:** toevoegen van een nieuwe interface aan een bestaande class

AOP + IoC: een unieke synergie

- AOP + IoC is een match made in heaven
- Elk object in de Spring IoC container can op een transparent manier geadviseerd (advised) worden gebaseerd op externe configuratie
- *Advisors, pointcuts* en *advices* kunnen zelf ook gemanaged worden door de IoC container
- *Spring is een consistente geïntegreerde oplossing voor IoC in combinatie met AOP*

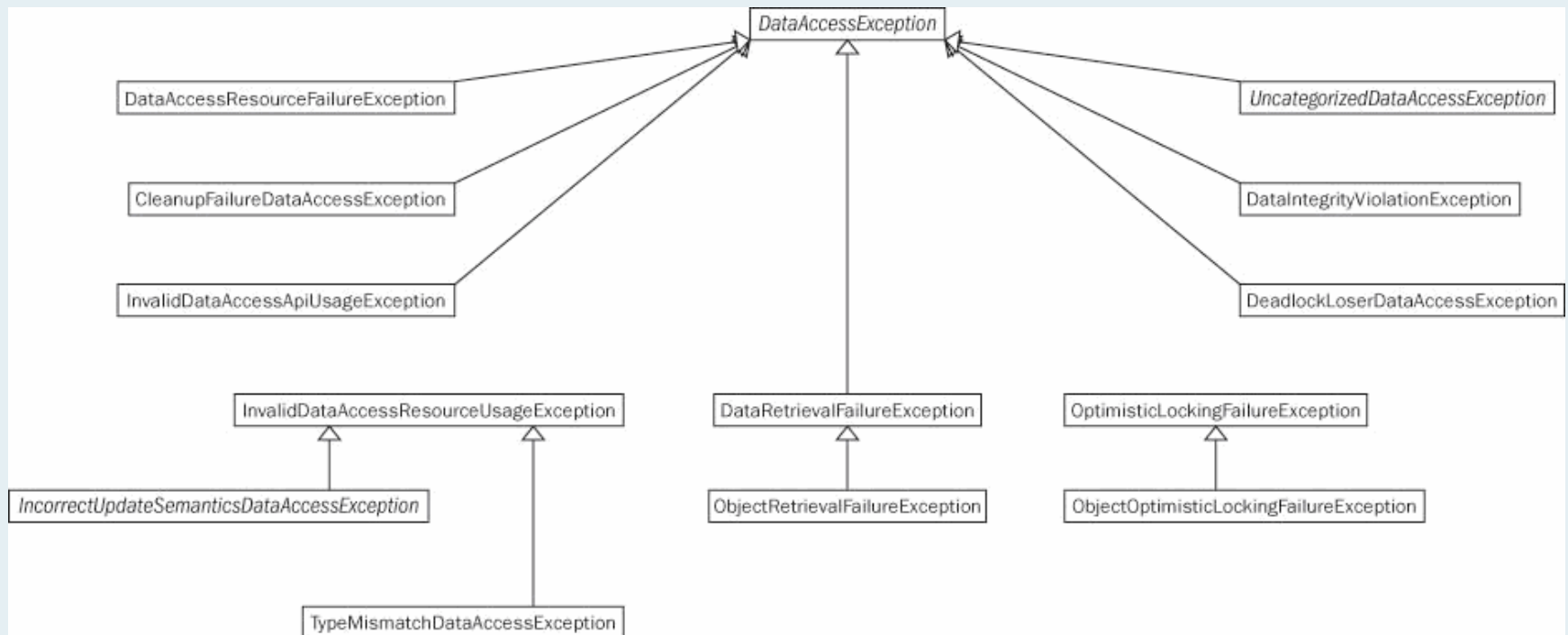
Custom AOP

- Conflicteert niet, maar complementeert OOP
 - Email administrator als een specifieke exceptie gegooid wordt
 - Voer custom security checks uit
 - Performance monitoring
 - Auditing
 - Caching
- Meer geavanceerd
 - Mixins via **introdunctie**
 - Voeg een `Lockable` interface toe aan een verzameling business objects
 - Mixin implementeert de `Lockable` methoden en andere methoden gedragen zich anders als het object gelockt is

Spring DAO

- Geïntegreerd met Spring Transaction Manager
- Vindt niet opnieuw het wiel uit
 - Er zijn goede oplossingen voor O/R mapping, wat Spring doet is deze gemakkelijker maken om te gebruiken
- Out-of-the-box support voor
 - JDBC
 - Hibernate
 - JDO
 - iBATIS
- Design staat andere technologieën toe (TopLink, etcetera)
- Gemeenschappelijk consistente DataAccessException staat volledig *technology-agnostic* DAOs toe

Spring DAO: Consistente Exception hierarchie



Spring DAO: JDBC

- Plain-old JDBC zonder class library is minder gemakkelijk
 - Twee manieren:
 - Callbacks (JdbcTemplate)
 - JDBC objecten: Modelleer queries, updates, stored procedures als object
- Geen try/catch/finally blocks meer!
- Geen lekkende connecties meer (maakt gebruik van onderliggende services)
- *Zinvolle exceptie hiërarchie*
 - Geen vendor specifiek code meer
 - Spring detecteert welke database je gebruikt en *weet wat ORA-098 betekent*
 - Portable code
 - Meer leesbare code
 - `catch (BadSqlGrammarException ex)`
- Store procedure support
- **Ideaal voor als een O/R mapper de functionaliteit niet biedt**

Spring DAO: Hibernate

- Beheert Hibernate session
 - Geen custom ThreadLocal of HTTP-based session object meer
 - Sessies worden beheerd ism met Spring's transaction management
 - Werkt *ook* samen met JTA waar nodig
 - Werkt *ook* samen met EJB container waar nodig
- HibernateTemplate maakt veelvuldig gebruikte operaties makkelijk
 - Simpele consistente exceptie afhandeling
 - Veel operaties zijn te vatten in slechts 1 regel code
- Consistentie exceptie hiërarchie
 - Switchen tussen Hibernate, JDO en andere persistence technologieën zonder de DAO interfaces te moeten aanpassen
 - Runtime exceptie
- Mix van Hibernate en JDBC *in dezelfde transactie* is zelfs mogelijk

HibernateTemplate DAO voorbeeld

```
public class MyHibernateDao implements MyDao {
    private HibernateTemplate hibernateTemplate;

    public MyHibernateDao (net.sf.hibernate.SessionFactory sessionFactory) {
        hibernateTemplate = new HibernateTemplate(sessionFactory);
    }

    public Collection getWorkflows() {
        return hibernateTemplate.find("from Workflow");
    }
}

<bean id="sessionFactory" class="org.springframework.orm.hibernate.LocalSessionFactoryBean">
    <property name="dataSource"><ref local="dataSource"/></property>
    <property name="mappingResources">
        <value>mycompany/mappings.hbm.xml</value>
    </property>
    <property name="hibernateProperties">
        <props>
            <prop key="hibernate.dialect">net.sf.hibernate.dialect.HSQLDialect</prop>
        </props>
    </property>
</bean>

<bean id="myDao" class="com.mycompany.MyHibernateDao"
    autowire="constructor"
>
```

Spring Transaction

- Consistente abstractie
 - PlatformTransactionManager
 - Vindt niet het wiel (TransactionManager) opnieuw uit
 - Switch tussen JTA, Hibernate, JDBC, local transactions door configuratie te veranderen *niet de Java code!*
 - Geen complete rewrite als een switch (omwille van schaalbaarheid) nodig is van local transacties naar JTA transacties
 - Grootst gemene deler; werkt altijd
- Programmatisch transacties manager
 - Gemakkelijker API dan JTA
 - Gebruik *dezelfde* API voor zowel Hibernate, JDBC, JTA, etcetera
 - *Write once have transaction management everywhere*TM ☺

Declarative Transaction Management

- Meest populaire keuze
- Gebruikt dezelfde *common ground* als programmatisch transaction management
- **Declaratief transacties managen voor elke POJO, zonder EJBs, zelfs zonder JTA**
- Flexibeler dan EJB CMT
 - Declarative rollback rules (MyCheckException)
 - **Non-invasive: minimaliseren van afhankelijkheden op Spring API**
 - Doorgeven van EJBContext is *niet meer nodig*

Laten we de ServiceImpl POJO transactioneel maken

```
public class ServiceImpl implements Service {
    private int timeout;
    private AccountDao accountDao;

    public void setTimeout(int timeout) {
        this.timeout = timeout;
    }

    public void setAccountDao(AccountDao accountDao) {
        this.accountDao = accountDao;
    }

    public void doSomething() throws ServiceWithdrawnException {
    }
}

<bean id="serviceTarget" class="com.mycompany.service.ServiceImpl">
    <property name="timeout"><value>30</value></property>
    <property name="accountDao"><ref local="accountDao"/></property>
</bean>
```

ServiceImpl transactioneel maken

```
<bean id="service"  
  class="org.springframework.transaction.interceptor.TransactionProxyFactoryBean"/>  
  <property name="target">  
    <ref local="serviceTarget"/>  
  </property>  
  <property name="transactionManager">  
    <ref local="localTransactionManager"/>  
  </property>  
  <property name="transactionAttributes">  
    <props>  
      <prop key="do*">  
        PROPAGATION_REQUIRED,-ServiceWithdrawnException  
      </prop>  
    </props>  
  </property>  
</bean>
```

ServiceImpl transactioneel maken

- Rollback regels voorkomen dat we `setRollbackOnly()` moeten aanroepen
 - Uiteraard kun je nog steeds programmatisch een transactie terug rollen
- Alternatieve aanpak in grotere applicaties:
 - Gebruik “auto proxy creator” om gelijksoortige attributen toe te passen op meerdere business objects (beans)
 - Gebruik metadata (Commons/JSR 175) of een andere pointcut aanpak om transactioneel gedrag op meerdere classes toe te passen
- Werkt met JTA, JDBC, Hibernate, JDO, iBATIS, JOTM transactions...
 - Alleen de definitie van de transactiemanager aanpassen

ServiceImpl transactioneel maken

```
/**
 * Echte commentaar hier
 *
 * @DefaultTransactionAttribute(TransactionDefinition.REQUIRES_NEW);
 * @RollbackRuleAttribute(ServiceWithdrawnException.class);
 */
public void doSomething(...) throws ServiceWithdrawnException {
}
```

- Als de configuratie eenmaal opgezet is, zal elk object met transaction attributes transactioneel worden wanneer verkregen via de Spring IoC container
- Transaction interceptor zal slechts methoden interceptor die transaction attributes hebben
- Het voorbeeld laat de Commons Attributes syntax zien
- JSR-175 support binnenkort beschikbaar (voor JDK1.5 final)

Spring OOP

- Geen custom singletons meer
 - Vaak gebruikt anti-pattern
- Programmeer mgv interfaces, niet classes
- Faciliteert het gebruik van het strategy pattern
 - Maakt goede OO practices gemakkelijker
- OO gedachte voorkomt dat IoC Dependency Injection rotzooit met je design en programmeer model
 - Baseer object granulariteit op OO concerns, niet op Spring concerns
- Combineer Spring met transparent persistence om een **echt domein model** te kunnen modelleren en coden

“Old” J2EE vs Spring

- Stateless Session Bean
 - Home interface
 - Component interface
 - “Business methods” interface
 - Bean implementatie class
 - Complexe XML configuratie
 - POJO delegate als je wil testen buiten de container

 - Veel van dit met als oorzaak EJB (afhankelijkheid)

 - *Parametrisatie is complex!*
 - *Lookups van environments variables (auw) of custom IoC framework*
- Spring Object
 - Business interface
 - Implementatie class
 - Straightforward XML configuratie

 - Eerste twee stappen moet je sowieso doen in Java

 - *Oeps, ik bedoelde eigenlijk Java-object en geen Spring object*

 - *Simpele configuratie of properties of afhankelijkheden is gemakkelijk*

“Old-style” J2EE vs Spring

- Stateless Session Bean
 - Service Locator, business delegate, JNDI lookup
 - Elke class die de service gebruikt is afhankelijk van de EJB interface (home.create)
 - *Moeilijk te testen buiten container*
- (Spring) Object
 - Logica in een plain-old-Java-object (POJO)
 - Afhankelijkheden en configuratie via setters en getters (of constructors)
 - Simpele, intuïtieve XML configuratie
 - Geen lookup code!
 - *Gemakkelijk te testen met mock objects buiten container*

Productiviteitsvoordeel

- Spring voorkomt onnodige code
- Geen Java-based glue-code meer maar een simpel XML formaat
 - Als Spring XML complex wordt, doe je waarschijnlijk dingen die vroeger met veel custom coding niet eens mogelijk waren
- Vroeger zou er veel custom Java coding en XML aan te pas komen
 - Deze XML was (is) geen standaard
- Combineer Spring met JDO of Hibernate en de productiviteitstoename is enorm

Performance

- Spring AOP-based declarative trans. Management performde 10-350% beter dan Stateless SessionBeans (afhankelijk van de EJB container)
- Stable/robuust
 - Lagere, meer consistente response times, ook onder zware load; zelfs in vergelijking met de high-end applicatie server die we gebruikten
- EJB is niet traag, Spring is snel en erg solide
- Zoekmachine prototype
 - Doel: 60tps; zonder moeite 150tps gehaald met Spring IoC, MVC en AOP
 - Uiteraard kreeg de consultancy die de proof-of-concept ontwikkelde de opdracht

Spring roadmap

- 1.0 uitgebracht March 2004 j.l.
- 1.0.1 uitgebracht April 22, 2004
 - Kleine verbeteringen en bugfixes
 - Focus op backward-compatibility
- 1.1 development op dit moment aan de gang
 - JMX, JMS support
 - Pointcut expression language
 - Dynamische herconfiguratie
 - Threadsafe
 - Object referenties blijven valide ondanks dat configuratie verandert
 - Ook hier: focus op backward-compatibility
 - Maar: een niet invasie framework dat jouw code loskoppelt van de onze, maakt het gemakkelijk om dit te handhaven
 - Ter vergelijking: EJB 2.1 → EJB 3.0
- Eclipse plugin, bruikbaar op dit moment
- “Rich client platform”
 - Spring niet alleen op het J2EE platform, ook op de client-side

...De Spring community

- Ten minste 4 boeken in 2004
 - *Spring Live* (May): Matt Raible
 - *J2EE Without EJB* (May): Johnson/Hoeller
 - *Professional Spring Development* (Q4): Johnson/Risberg/Hoeller/Arendsen
 - Bruce Tate's nieuwe boek (Juni)
- Gerelateerde projecten
 - Acegi Security voor Spring

Wie gebruikt Spring

- Banking
 - Globale investeringsbank: 2 projecten live met Spring MVC IoC, JDBC, AOP; 10.000 gebruikers dagelijks
 - Duitse bank
- Verschillende overheden en onderwijs instellingen
 - Europese commissie
 - WHO
 - CERN
 - Universiteiten in US en UK
 - Rutgers University (New Jersey)
 - Warwick University (UK)
- Nederland: Ilse Media (offerte management systeem)

Spring

j2ee Application Framework

Bedankt voor de aandacht!
tot ziens op
www.springframework.org

Alef Arendsen (alef@jteam.nl)

JTeam B.V. - Amsterdam