

INTERFACE21



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK


Anwendungsentwicklung mit Spring

Eberhard Wolff
Managing Director
Interface21 GmbH
Interface21 – Spring from the Source

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.




INTERFACE21



Überblick

- Über Interface21
- Integration von Frameworks in Spring
- Komponenten und Objekte
- Separation of Concerns und aspektorientierte Programmierung (AOP)
- Architektur

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Interface21

- Produkte u.a. Spring Framework
- Spring from the Source
- Interface21 beschäftigt alle Spring-Committer
- Consulting, Training, Support
- Kunden: u.a. 6 der 10 weltweit größten Banken
- Mitarbeiter:
 - Rod Johnson CEO (Erfinder Spring)
 - Jürgen Höller Distinguished Engineer (Chef Entwickler Spring)
 - Adrian Colyer CTO (Kopf hinter AspectJ)
- 10 Mio \$ Investment von Benchmark (Venture Capitalist hinter MySQL, Red Hat etc.)

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.




Über mich

- Managing Director Interface21 Deutschland
- Java Champion
- Fokus: Java EE, Spring ...
- Autor (z.B. Java Magazin, 3 Bücher...)
 - Z.B. Spring
- Blog: <http://JandlandMe.blogspot.com/>

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.




INTERFACE21




ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Integration von Frameworks in Spring

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



INTERFACE21



Das Problem vieler Java-APIs

```
Connection con = null;
ResultSet rs = null;
int result = 0;
try {
    con = ...;
    Statement stmt = con.createStatement();
    rs = stmt.executeQuery("SELECT count(*) FROM KUNDE");
    if (rs.next()) {
        result=rs.getInt(1);
    }
    stmt.close();
} catch (SQLException e) {
    System.out.println("SQL-Exception:"+e);
} finally {
    con.close();
}
```

Größtenteils schwer diagnostizierbar!

rs.close() fehlt (?)


Wird nicht immer ausgeführt

Fehlerbehandlung?

NullPointerException?

SQLException?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Richtig: Try Catch Finally Try Catch

```
Connection con = null; Statement stmt = null; ResultSet rs = null;
try {
    con = ...;
    Statement stmt = con.createStatement();
    ...
} catch (SQLException e) {
    System.out.println("SQL-Exception:"+e);
} finally {
    if (stmt!=null) {
        try { stmt.close();
        } catch (SQLException ex) { }
    }
    if (con!=null) {
        try { con.close();
        } catch (SQLException ex) { }
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Das kann man
keinem
Entwickler
zumuten.

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Das kann man keinem Entwickler zumuten

- „Benutzen Sie JDBC?“
- „Ja.“
- „Sie haben ein Problem.“
- Oder auch:
- „Wir haben Garbage Collection, aber was ist mit den Ressourcen?“
- Oder auch:
- Was ist das JDBC-Tutorial verschweigt...

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Probleme

- APIs sind *Standards*, Benutzbarkeit nicht im Vordergrund
- Exception Handling: ein kompliziertes Thema...
- ... klar ist: technische Exceptions sollten `RuntimeExceptions` aka `unchecked Exceptions` sein
- Andere Sprachen verwenden nur `unchecked Exceptions`
- Können wie `NullPointerExceptions` überall vorkommen
- Ressourcen Handling ist problematisch

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Dasselbe mit Spring 2.0

```

Connection con = null; Statement stmt = null; ResultSet rs = null;
try {
    con = ...;
    Statement stmt = con.createStatement();
    ...public int getNumber() {
} catch (SQLException ex) {
} return getSimpleJdbcTemplate().queryForInt(
    System.out.println("COUNT=" + rs.getInt(1));
    "SELECT COUNT(*) FROM KUNDE");
} finally {
    if (stmt != null) {
        try { stmt.close();
        } catch (SQLException ex) { }
    }
    if (con != null) {
        try { con.close();
        } catch (SQLException ex) { }
    }
}
}

```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Komplexeres Beispiel

```

// Callback macht aus einem ResultSet einen Kunden
private static class KundeResultSetRowMapper
    implements ParameterizedRowMapper<Kunde> {
    public Kunde mapRow(ResultSet rs, int rowNum) {
        return new Kunde(rs.getString(1)...);
    }
}

public List<Kunde> getByName(String name) {
    return getSimpleJdbcTemplate().query(
        "SELECT * FROM KUNDE WHERE NAME=?",
        new KundeResultSetRowMapper(), name);
}

```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Template Pattern

- Problem: Aufräumen der Ressourcen vor allem bei Exceptions
- Lösung: Auszuführenden Code einem Template übergeben
 - Template führt Code aus
 - ...und räumt Ressourcen auf
- Templates sind in Spring für viele APIs implementiert
 - Hibernate, iBATIS, JDO, JPA (EJB 3 Persistenz), JCA, JNDI, JMS...
- Einheitliche Nutzung verschiedener APIs

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Exception Übersetzer

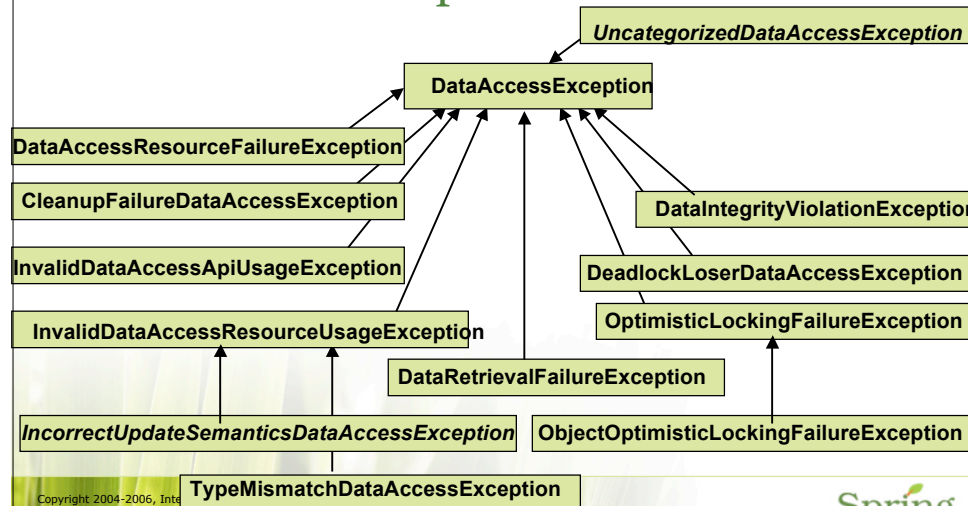
- Problem: `SQLException`
 - Gibt proprietäre Codes zurück
 - Ist eine checked Exception (keine `RuntimeException`), obwohl man selten eine sinnvollen Reaktionsmöglichkeiten hat
- Lösung: `JdbcTemplate` implementiert auch Exception Übersetzer Pattern
 - Wandelt Exceptions in `RuntimeExceptions` um
 - Gemeinsame Exception Hierarchie für alle Datenbanken
- Exception Übersetzer für viele technische APIs implementiert

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Ausschnitt aus der DataAccessException Hierarchie



Hilfsklassen: JDBC Komplexere Queries

- Für den Extremfall gibt es Möglichkeiten auf Statement / PreparedStatement Ebene
- ...und auch mit OraclePreparedStatement
- Also: Ebenfalls Möglichkeiten für komplexe JDBC-Programmierung
- Man verliert nie die Vorteile des Template und Exception Übersetzer Patterns



Anderes Beispiel: JMS

```
QueueConnectionFactory qcf = ...;
QueueConnection qc = null;
Queue queue = null;
try {
    qc = qcf.createQueueConnection();
    QueueSession qsession = qc.createQueueSession(
        false, Session.AUTO_ACKNOWLEDGE);
    QueueSender qsender= qsession.createSender(queue);
    TextMessage msg = qsession.createTextMessage();
    msg.setText("Text");
    qsender.send(msg);
} catch (JMSEException jmsex) {
    jmsex.printStackTrace();
} finally {
    qc.close();
}
```

} Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Das kann man
keinem
Entwickler
zumuten.

} Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





JMS mit Spring

```
jmsTemplate.convertAndSend("Eine Text Message");
```

Sorry, das ich auf dieser Folie
soviel Platz verschenkt habe!



JMS mit Spring

oder:

```
jmsTemplate.send(new MessageCreator() {  
    public Message createMessage(Session session)  
        throws JMSException {  
        return session.createTextMessage("Text Message");  
    }  
});
```



Bemerkung

- Diese durchgehende API-Abstraktion gibt es nur in Spring!
- Macht das Leben mit vielen APIs einfacher
- Getrennt vom Rest von Spring nutzbar
- Durchgehendes, vereinfachtes *Programmiermodell*
- Niemand behauptete, Java (EE) sei eine schlechte Infrastruktur!

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Und?

- Die Nutzung der APIs wird wesentlich vereinfacht
- Wie soll ich meine Anwendung strukturieren?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring


INTERFACE21 

ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Objekte und Komponenten

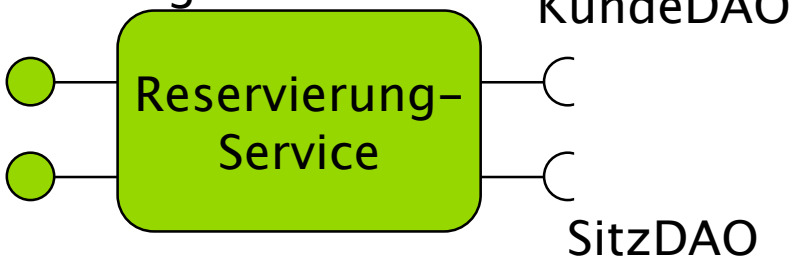
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21 

Komponenten

Reservierung



```
graph LR; RS[Reservierung-Service]; K[KundeDAO]; S[SitzDAO]; RS --> K; RS --> S;
```

The diagram shows a central green rounded rectangle labeled "Reservierung-Service". To its left, two green circles are connected to the rectangle by horizontal lines. To its right, two semi-circular "provided interface" symbols are connected to the rectangle by horizontal lines. The top semi-circle is labeled "KundeDAO" and the bottom one is labeled "SitzDAO".

Merkmal einer Komponente:
Schnittstellen und Abhängigkeiten sind von außen erkennbar

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Wie einfache können Komponenten sein?

- Schnittstelle = normales Java Interface

```
public class ReservierungServiceImpl
    implements ReservierungService {

    public void reserviere(Sitz sitz) {
        ...
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Von außen nicht erkennbar Abhängigkeiten: EJB 2/ Java EE / JNDI

- Java EE verwendet JNDI (Java Naming and Directory Interface) als Namenssystem
- Die Abhängigkeiten sind von außen nicht erkennbar, sondern im Code „versteckt“
- Was muss die Komponente in der Umgebung finden, um zu funktionieren?

```
InitialContext initialContext = new InitialContext();
Object objRef = initialContext.lookup("KundeDAO");
KundeDAOHome lookupHome =
    (KundeDAOHome) PortableRemoteObject
        .narrow(objRef, KundeDAOHome.class);
initialContext.close();
KundeDAO bean = lookupHome.create();
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Von außen nicht erkennbar Abhängigkeiten: Factory und Singleton

- Factory Pattern: Erzeugung in Klasse auslagern
- Singleton Pattern: Genau eine Instanz erzeugen und vorhalten
- Von außen ist die Verwendung von `MyFactory` und `SitzDAO` nicht zu erkennen

```
public void zuTestendeMethode() {  
    KundeDAO kundeDAO = MyFactory.createKundeDAO();  
    SitzDAO sitzDAO = SitzDAO.getInstance();  
    // DAOs nutzen  
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Von außen erkennbare Abhängigkeiten mit Java

```
public class ReservierungServiceImpl  
    implements ReservierungService {  
  
    private KundeDAO kundeDAO;  
    private SitzDAO sitzDAO;  
  
    public void setKundeDAO(KundeDAO kundeDAO) {  
        this.kundeDAO = kundeDAO;  
    }  
    public void setSitzDAO(SitzDAO sitzDAO) {  
        this.sitzDAO = sitzDAO;  
    }  
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Alternative Konstruktoren

```
public class ReservierungServiceImpl
    implements ReservierungService {

    private KundeDAO kundeDAO;
    private SitzDAO sitzDAO;

    public ReservierungServiceImpl(KundeDAO kundeDAO,
        SitzDAO sitzDAO) {
        this.kundeDAO = kundeDAO;
        this.sitzDAO = sitzDAO;
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Abhängigkeiten grafisch



Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Und nun?

- Das ganze heißt Dependency Injection (DI) oder Inversion of Control (IoC)
- Eine Art Pattern: Man kann es auf viele unterschiedliche Weisen implementieren
- Aber: Irgendwie müssen die Komponenten zusammengesteckt werden.



Eine Möglichkeit: Coden wir's doch!

```
ReservierungImpl component =  
    new ReservierungImpl();  
  
component.setKundeDAO(  
    MyFactory.createKundeDAO());  
component.setSitzDAO (  
    SitzDAO.getInstance());  
  
component.doSomething();
```




Coden?

- Langweiliger, sich ständig wiederholender Code
- Man sieht den Wald vor lauter Bäumen nicht
- Kann man nicht eine Infrastruktur erfinden?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Infrastruktur: Spring

```
<beans>
  <bean id="sitzDAO"
    class="SitzDAO"
    factory-method="getInstance" />
  <bean id="kundeDAO"
    class="MyFactory"
    factory-method="createKundeDAO" />
  <bean id="component" class="MyComponent">
    <property name="sitzDAO"
      ref="sitzDAO" />
    <property name="kundeDAO"
      ref="kundeDAO" />
  </bean>
</beans>
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Bemerkung Spring

- Wie man sieht sind auch Factories möglich
- Konstruktor-Parameter (wie diskutiert) auch
- Andere DI-Lösungen bieten diese Flexibilität nicht
- Dadurch keine Möglichkeit für die Integration von anderen Libraries und Legacy-Code
- Oft hängt dann noch der Code von der DI-Lösung ab
- Bean werden per default beim Start initialisiert
- ...und zwar als Singleton
- So kann zum Beispiel ein GUI-Hauptfenster initialisiert werden

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Beispiel Code

```
public static void main(String[] args) {  
    ApplicationContext applicationContext =  
        new ClassPathXmlApplicationContext  
            ("beans.xml");  
    MyComponent testObjekt = (MyComponent)  
        applicationContext.getBean("testObjekt");  
    testObjekt.someCalculation();  
}
```

Auslesen der Objekte ist allgemein nicht notwendig

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Beispiel GUI

```
<beans>
```

```
  <bean class="javax.swing.JFrame">
    <property name="visible" value="true" />
    <property name="title" value="Titel" />
  </bean>
```

```
</beans>
```

Logik kann in GUI injiziert werden.

Ähnliches ist vorhanden für Testfälle (JUnit),

Web Frameworks etc.

Dependency Injection geht durch die ganze Anwendung

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Web Konfiguration

```
<web-app>
```

```
  <context-param>
```

```
    <param-name>contextConfigLocation</param-name>
```

```
    <param-value>
```

```
      /WEB-INF/data-access.xml
```

```
      /WEB-INF/business-layer.xml
```

```
    </param-value>
```

```
  </context-param>
```

```
  <listener>
```

```
    <listener-class>
```

```
      org.springframework.web.context.ContextLoaderListener
```

```
    </listener-class>
```

```
  </listener>
```

```
</web-app>
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Aber...

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Das ist ja XML!



Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Aber das ist ja XML

- Einfaches XML
- Mit dem Anspruch, lesbar und leicht editierbar zu sein
- Mit Tool-Support (Spring IDE)
- Aber: Man kann auch stattdessen Java & Annotationen verwenden
- Neues Projekt: Spring JavaConfig

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Spring JavaConfig

```
@Configuration
public class SpringConfiguration {
    @Bean
    private SitzDAO sitzDAO() {
        return SitzDAO.getInstance();
    }
    @Bean
    private KundeDAO kundeDAO() {
        return MyFactory.createKundeDAO();
    }
    @Bean
    public ReservierungService component() {
        ReservierungServiceImpl component =
            new ReservierungServiceImpl();
        component.setSitzDAO(sitzDAO());
        component.setKundeDAO(kundeDAO());
        return component;
    }
}
```

- Von den Beans gibt es jeweils nur eine Instanz
- Mit XML kombinierbar
- Der Kern des Spring Dependency Injection Container ist unabhängig von der Konfiguration

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Neu in Spring 2.5: DI mit Annotationen in den Spring-Bean-Klassen

```
@Repository
public class StubAccountRepository
    implements AccountRepository {
}

public class ConstructorAutowiredTransferService{
    @Autowired
    public ConstructorAutowiredTransferService(
        AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Vorteil: Unabhängiges Deployment

- Klassen können einzeln deployt werden
- ...funktionieren aber natürlich nur, wenn eine sinnvolle Umgebung vorhanden ist
- Anforderungen an die Umgebung sind durch explizite Abhängigkeiten klar

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Vorteil: Komposition

- Durch Dependency Injection sind die Komponenten unabhängig von der Umgebung
- ...sie kennen die Umgebung noch nicht einmal
- ...die besteht aus den injekteten Klassen
- Klasse kann also mit andere zusammenspielen



Komposition

```
<beans>
  <bean id="sitzDAO" class="AnderesSitzDAO"
    factory-method="getInstance" />
  <bean id="kundeDAO" class="DifferentFactory"
    factory-method="createKundeDAO" />
  <bean id="component"
    class="ReservierungsServiceImpl">
    <property name="sitzDAO"
      ref="sitzDAO" />
    <property name="kundeDAO"
      ref="kundeDAO" />
  </bean>
</beans>
```



Unit-Tests sind ein Sonderfall der Komposition

- Definition Unit-Test: Klasse in *Isolation* testen
- Also keine abhängigen Klassen mittesten
- ...sondern stattdessen Dummies / Mocks
- Nach Link:
 - Dummy ersetzt die eigentliche Klasse für Tests durch eine andere Implementierung ersetzt, um ein bestimmtes Verhalten zu simulieren.
 - Mock: Dummy und man kann das Verhalten konfigurieren kann.
- Komponiere die Komponente so, dass sie Mocks verwendet!
- Kann man, dass Abhängigkeiten explizit sind

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Beispiel Unit-Test

```
public class MyTestCase extends TestCase {  
  
    public void testUnitTest() {  
        ReservierungDAOImpl testComponent =  
            new ReservierungDAOImpl();  
        testComponent.setKundeDAO(new KundeDAODummy());  
        testComponent.setSitzDAO(new SitzDAODummy());  
  
        testComponent.zuTestendeMethode();  
        assertEquals(21, testComponent.getWert());  
    }  
}
```

Alternative: Zum Beispiel Mocks mit Easymock

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Dependency Injection

- Verbessert Reuse und Testbarkeit
- Spring bietet eine umfangreiche Unterstützung vieler verschiedener DI-Mechanismen
- Der Spring-DI-Kern ist leicht erweiterbar

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Aufrufe über das Netz

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Aufrufe über das Netzwerk

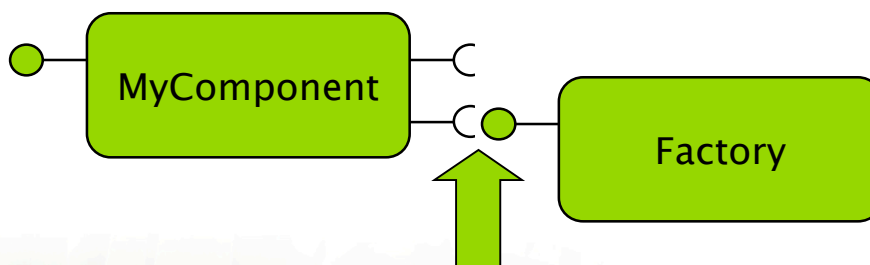
- In EJB 3 und Java EE 5 durch Annotationen und inhärent im Komponenten-Modell
- Dadurch enthält der Code Hinweise darauf, wie etwas verteilt aufgerufen werden kann.
- Was, wenn eine neue Technologie auftaucht?
- Was, wenn ich auch noch eine SOAP-Schnittstelle will?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Abhängigkeiten grafisch



Verbindung wird durch Dependency Injection Container erzeugt, also indirekt

Kann man da auch ein Netzwerk zwischen bekommen?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Alternative: Spring Exporter und Proxy

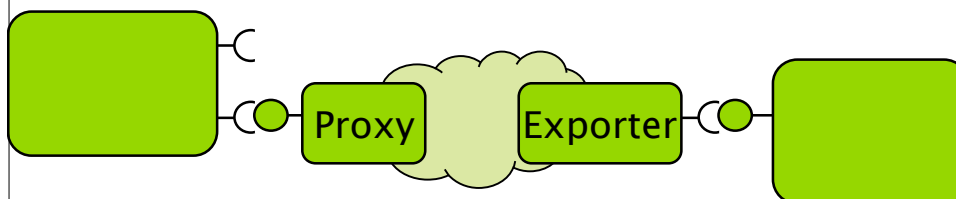
- Ein Exporter exportiert eine Spring-Bean mit einer passenden Technologie
- Umgekehrt wird mit einem Proxy Aufrufe über das Netz an einen Server weitergegeben
- Beispiel: HttpInvoker (HTTP+Serialisierung)
- Bemerkung: HTTP ist ein einfaches, ausfallsicheres und skalierbares Protokoll

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Exporter / Proxy grafisch



Proxy und Exporter “reden” über irgendein Protokoll miteinander

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



HttpInvoker: Exporter auf dem Server

```
<bean id="myComponentExporter"  
  class="...HttpInvokerServiceExporter">  
  <property name="service" ref="myComponent" />  
  <property name="serviceInterface"  
    value="SomeInterface" />  
</bean>
```

Referenziert per Dependency Injection eigentliche
Komponente
...die nichts davon merken, dass sie ein Server sind

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring



HttpInvoker: Proxy auf dem Client

```
<bean id="kundeDAOHttpInvoker"  
  class="...HttpInvokerProxyFactoryBean">  
  <property name="serviceInterface"  
    value="SomeInterface" />  
  <property name="serviceUrl"  
    value="http://localhost:8080/some/myComponent" />  
</bean>
```

Wird per Dependency Injection anderen Komponenten
injiziert
...die nichts davon merken, dass sie Clients sind

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or
distributing without expressed written permission is prohibited.

Spring



Ergebnis Proxy / Exporter

- Eigentlicher Code vollkommen unabhängig von der Verteilung!
- Danke XFire auch für SOAP
- Bei EJB 2 leider Coding notwendig
- Nicht nur Verteilung sondern auch Management per JMX
- ...oder OSGi Container
- Eigentlicher Code technologieutral
 - Einfacheres Testen: Ohne Infrastruktur
 - Investitionsschutz: Sie sind auch auf den nächsten Kommunikationsstandard vorbereitet
 - Andere Infrastruktur leicht nutzbar falls notwendig

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Und was ist mit anderen Dingen?

- Transaktionen
- Security
- Monitoring
- Tracing
- ...

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Separation of Concerns und aspektororientierte Programmierung (AOP)

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



INTERFACE21



Ziel: Separation of Concerns

- Dijkstra 1974
- Idee: Man will über unterschiedliche Belange getrennt nachdenken können
- Erst baut man Geschäftslogik und will nur diese sehen...
- ...dann Security und man will nur Security sehen
- Variation: Do it once and only once.
- Geht das?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Sind hier die Belange separiert?

```
public String getSomeInformation ()
throws AnException {
    final String METHOD_NAME = "getSomeInformation ";
    Trace.enterScope(CLASS_NAME, METHOD_NAME);
    if (youAreNotAllowedToDoThis) {
        throw new SecurityException();
    }
    try {
        ...
    } catch (AnException ex) {
        Transaction.current().rollback();
    } finally {
        Trace.exitScope();
    }
    return aResult;
}
```

Tracing**Security****Transaktionen**

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Cross Cutting Concerns

- Java bietet die hierarchische Aufteilung eines Systems in Packages, Klassen, Methoden
- Kann man Security usw separiert an einer Stelle implementieren?
- Nein!
- Klassische Modularisierung reicht nicht
- Also fehlt irgendwas.

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Was, wenn man?

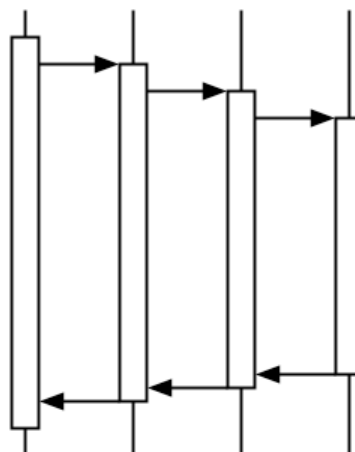
- ...einen Methodenaufruf abfangen kann und noch etwas zusätzliches tun kann?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Methoden „abfangen“



Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Aufrufendes Spring Method Eigentliche
Objekt Objekt Interceptor Methode

Spring



Was wir benötigen: Advice

- Advice: Was ausgeführt wird
- Eine Stück normaler Java-Code
- Z.B. Around-Advice: Legt sich um den eigentlichen Code „herum“ und kann ihn aufrufen
- ...oder auch nicht (Caching)
- Sonst: Before-Advice usw.

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Was wir benötigen: Pointcut

- Pointcut: Wo es ausgeführt werden soll
- Definition der zu erweiternden Methoden
- Genau eine spezifische Methode in einer beliebigen Klasse:
`execution(void eineMethode())`
- Wildcard für Rückgabewert, Klasse und Parameter:
`execution(* aop.Test.*(..))`

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Beispiel: Retry

Methode bei Exception nochmal aufrufen

```
@Aspect
public class RetryInterceptor {
    @Around("execution(* aop.Test.*(..))")
    public Object retry(ProceedingJoinPoint proceedingJoinPoint)
        throws Throwable {
        try {
            return proceedingJoinPoint.proceed();
        } catch (Throwable ex) {
            return proceedingJoinPoint.proceed();
        }
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Es geht auch ohne Annotationen...

```
<aop:config proxy-target-class="true">
  <aop:aspect ref="interceptor">
    <aop:around method="retry"
      pointcut="execution (* aopconfig.Test.*(..))" />
  </aop:aspect>
</aop:config>
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Wie ist das implementiert?

- Spring: Dynamic Proxy fängt Aufruf ab
- ...bzw. die Klasse wird zur Laufzeit durch eine Subklasse mit CGLIB ersetzt
- Die @Aspect-Schreibweise kann auch AspectJ auswerten
- AspectJ hat einen eigenen Compiler
- ...oder bietet Modifikation zur Laufzeit (Load Time Weaving)

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



AOP ist ein Enabler

- Mit AOP kann man Annotationen Semantik geben.
- Beispiel: Security mit Acegi
- Anderes Beispiel: Transaktionen für Spring
- Man benutzt AOP, ohne dass man es merkt

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Annotationen & AOP: Beispiel Spring Security Acegi

```
public interface BankManager {  
  
    @Secured({"ROLE_SUPERVISOR","RUN_AS_SERVER" })  
    public void deleteSomething(int id);  
  
}
```

Acegi Interceptor unterbindet ggf. Zugriff
Acegi kann viele Sicherheitsinfrastrukturen integrieren



Annotationen & AOP: Spring Transaktionen

```
@Transactional(propagation=Propagation.REQUIRED)  
public void bestellen(  
    Einkaufswagen einkaufswagen,  
    int kreditkartenNummer)  
    throws BestellungException {  
    ...  
}
```



Wie baue ich selbst so einen Pointcut?

```
<aop:config proxy-target-class="true">
  <aop:aspect ref="interceptor">
    <aop:around method="trace"
      pointcut=
        "execution(@annotation.Retry * *.*(..))" />
    </aop:aspect>
  </aop:config>
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

Architektur

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



And now for something completely different: Architektur

- Aufteilung einer Anwendung verschiedene Teile
- Können wir dafür nicht Pointcuts definieren?
- Und was können wir dann tun?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Pointcuts für Architektur

```
@Aspect
public class SystemArchitektur {
    @Pointcut("within(service.*)")
    public void inServiceLayer() {}
    @Pointcut("within(dao.*)")
    public void inDAOLayer() {}

    @Pointcut("execution(* service.*(..))")
    public void executionServiceLayer() {}
    @Pointcut("execution(* dao.*(..))")
    public void executionDAOLayer() {}
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





RuntimeExceptions in der Service-Schicht müssen geloggt werden

```
@Aspect
public class ExceptionHandling {

    @AfterThrowing(throwing="ex",
        pointcut=
            "architektur.SystemArchitektur.executionServiceLayer()")
    public void logRuntimeException(RuntimeException ex) {
        System.out.println("Something bad happend: "+ex);
    }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



Damit können ganze Layer eingespart werden

- Gerade auf dem Service-Layer setzten oft noch Adapter auf
- Exception Behandlung
- Mitschneiden der Methode für Wiederaufsetzen z.B. als Commands
- Security
- Transactions

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Mehr Power Point Architektur

- „Die Benutzung von JDBC ist nur im DAO Layer erlaubt.“
- „Exceptions müssen geloggt werden. Aufrufe von `printStackTrace()` sind nicht zulässig.“

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.



...und Software Artefakt (AspectJ)

```
@DeclareError(" (call(* java.sql.*.*(..)) && " +  
    "!within(*.dao.*) ) ")  
public static final String JdbcOnlyInDAOs =  
    "JDBC only in DAOs!";
```

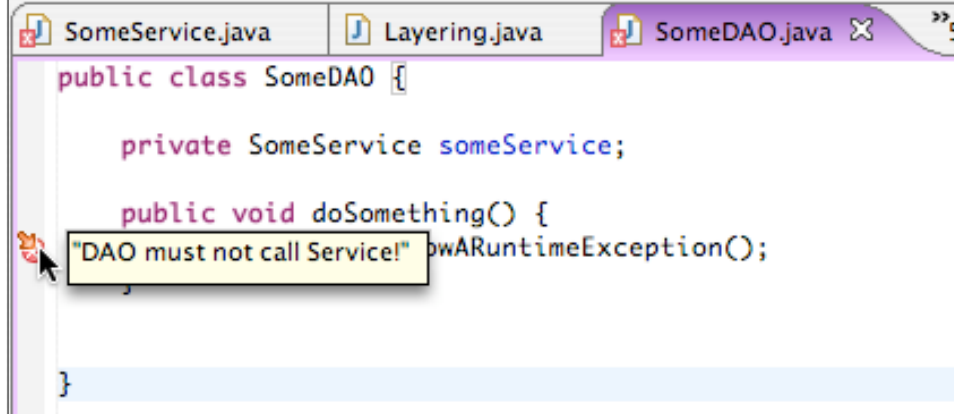
```
@DeclareError("call(void " +  
    "java.lang.Throwable+.printStackTrace())")  
public static final String NoPrintStackTrace = "Please  
log exception!";
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.





Das werden Compiler Fehler!



Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Zusammenfassung



- Spring bietet eine durchgehende Vereinfachung für praktisch alle Java APIs
- Einzigartig
- Unabhängig vom Rest von Spring nutzbar
- Implementierte Patterns: Template / Exception Übersetzer

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Zusammenfassung



- Mit Dependency Injection und Java-Bordmitteln kann man ein Komponentenmodell aufbauen
- Vorteil: Testbarkeit, Komponierbarkeit, unterschiedliche Infrastrukturen (z.B. SOAP) nutzbar (Exporter / Proxies)
- Investitionsschutz: Geschäftslogik technologieunabhängig und damit gegenüber Technologie-Änderungen stabil

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Zusammenfassung



- AOP ermöglicht weiteren Separation of Concern z.B. für Transaktionen oder Sicherheit
- Damit kann man Architektur definieren
- ...und auch die Umsetzung der Architektur forcieren

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Andere Dinge...

- Spring Dynamic Modules for OSGi™ Platform
- Spring Web Flow
- Spring Web Services
- Spring Security (Acegi)
- Spring JavaConfig
- Spring IDE / Tool Suite
- Pitchfork (EJB 3 auf Spring)...
- Spring ist “nur” der Anfang

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Mein Buch

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Und natürlich...

- ...helfen wir Ihnen gerne!
- Interface21 – Spring from the Source!
- Training – Consulting – Support
- Nächstes öffentliches Training:
20.–23. November in Stuttgart

Eberhard.Wolff@interface21.com