7 (or so) Reasons to use Spring

Arjen Poutsma
SpringSource
VMware
About Spring
Aims of Spring

• Reduce the complexity of Java J2EE development

• Simplify without sacrificing power

• Facilitate best practices

• Grew from practical experience

Simple things should be simple. Complex things should be possible.

Alan Kay
Technical Aims of Spring

- Write POJOs
- Apply Java EE services to POJOs
  - Make it transactional
  - Expose via JMX
  - ...

woensdag 6 oktober 2010
POJO development

- Plain Old Java Object
- Not bound to any environment
  - No environment-specific imports
  - Not dependent on lookup mechanism
    - Dependencies are injected
  - Prolongs life
- Test out of the container
Modular

• Spring is not a “package deal”
• All features can be used independently
  • Though they strengthen each other
Reason #1
Dependency Injection
The Case for Dependency Injection

- Applications consist of multiple components
- How obtain these dependencies?
Pull Configuration

• JNDI lookup
• Properties file
• Service Locator anti-pattern
• Many issues
  • Most importantly: hard to test
Solution: Dependency Injection

- Rather than lookup dependencies
- Let something give them to me
- Spring
How Spring Works

Spring Application Context
How Spring Works

Your Application Classes (POJOs)

Spring Application Context
How Spring Works

Your Application Classes (POJOs)

Configuration Instructions

Spring Application Context
How Spring Works

Your Application Classes (POJOs)

Configuration Instructions

Spring Application Context

Creates

Fully configured application system

Ready for use
Example
Example

```java
public class TransferServiceImpl implements TransferService {
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    ...
}
```
public class TransferServiceImpl implements TransferService {
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    …
}

Needed to perform money transfers between accounts

Example

woensdag 6 oktober 2010
Example

```java
public class TransferServiceImpl implements TransferService {
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    ...
}
```

```java
public class JdbcAccountRepository implements AccountRepository {
    public JdbcAccountRepository(DataSource ds) {
        this.dataSource = ds;
    }
    ...
}
```

Needed to perform money transfers between accounts

woensdag 6 oktober 2010
Example

```java
public class TransferServiceImpl implements TransferService {
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    ...
}
```

Needed to perform money transfers between accounts

```java
public class JdbcAccountRepository implements AccountRepository {
    public JdbcAccountRepository(DataSource ds) {
        this.dataSource = ds;
    }
    ...
}
```

Needed to load accounts from the database

woensdag 6 oktober 2010
Example
<beans>

  <bean id="transferService" class="app.impl.TransferServiceImpl">
    <constructor-arg ref="accountRepository" />
  </bean>

  <bean id="accountRepository" class="app.impl.JdbcAccountRepository">
    <constructor-arg ref="dataSource" />
  </bean>

  <bean id="dataSource" class="com.oracle.jdbc.pool.OracleDataSource">
    <property name="URL" value="jdbc:oracle:thin:@localhost:1521:BANK" />
    <property name="user" value="moneytransfer-app" />
  </bean>

</beans>
“Spring Sucks!”

- Spring is XML
- XML is Evil
- Being Evil sucks

- Therefore, Spring sucks
Spring != XML

• @Component
• @Autowired
• JSR-250
  • @Resource
• @PostConstruct/@PreDestroy
• @Configuration
@Component
@Component
@Autowired
public class TransferServiceImpl implements TransferService
    @Autowired
    public TransferServiceImpl(AccountRepository ar) {
        this.accountRepository = ar;
    }
    ...
}
Why not use Java EE Dependency Injection?

- Testing, inside IDE
- Java EE only allows injection of JNDI-managed objects
- Spring injects everything
  - Primitives (configuration)
Reason #2
JdbcTemplate
JDBC

- Object/Relational Mapping is popular
- But JDBC continues to be important
  - Batch Operations
  - Set-based operations
  - Stored procedures
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
```java
int count = jdbcTemplate.queryForInt(
    "SELECT COUNT(*) FROM CUSTOMER");
```
int count = jdbcTemplate.queryForInt("SELECT COUNT(*) FROM CUSTOMER");

- Acquisition of the connection
- Participation in the transaction
- Execution of the statement
- Processing of the result set
- Handling any exceptions
- Release of the connection

All handled by Spring
Querying With SimpleJdbcTemplate

```java
public int getCountOfPersonsOlderThan(int age) {
    return jdbcTemplate().queryForInt(
        "select count(*) from PERSON where age > ?", age);
}
```
Domain Objects

```java
public Person getPerson(int id) {
    return getSimpleJdbcTemplate().queryForObject(
            "select first_name, last_name from PERSON where id=?",
            new PersonMapper(), id);
}
```
public Person getPerson(int id) {
    return getSimpleJdbcTemplate().queryForObject(
            "select first_name, last_name from PERSON where id=?",
            new PersonMapper(), id);
}

class PersonMapper implements ParameterizedRowMapper<Person> {
    public Person mapRow(ResultSet rs, int i) {
        return new Person(rs.getString(1), rs.getString(2));
    }
}

woensdag 6 oktober 2010
Reason #3
Exception Hierarchy
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
public List findByLastName(String lastName) {
    List personList = new ArrayList();
    Connection conn = null;
    String sql = "select first_name, age from PERSON where last_name=?";
    try {
        DataSource dataSource = DataSourceUtils.getDataSource();
        conn = dataSource.getConnection();
        PreparedStatement ps = conn.prepareStatement(sql);
        ps.setString(1, lastName);
        ResultSet rs = ps.executeQuery();
        while (rs.next()) {
            String firstName = rs.getString("first_name");
            int age = rs.getInt("age");
            personList.add(new Person(firstName, lastName, age));
        }
    } catch (Exception e) { /* ??? */ }
    finally {
        try {
            conn.close();
        } catch (SQLException e) { /* ??? */ }
    }
    return personList;
}
What can you do?

- Nothing
- Logging
- Wrapping
- Retry
Dead Lock Loser

```java
} catch (SQLException ex) {
    if (ex.getErrorCode() == 60) { // check for ORA-00060
        return findByLastName(lastName);
    } else {
        throw ex;
    }
}
```
Retry

- Ugly code
- Not portable
  - JDBC vs JPA
  - Oracle vs SQL Server
In Spring

- Spring does exception translation
- Into a rich Exception Hierarchy
  - Catch DeadLockLoserException
- Use AOP!
DataAccessException
Hierarchy (subset)

- DataAccessException
  - DataAccessResourceFailureException
  - CleanupFailureDataAccessException
  - InvalidDataAccessApiUsageException
  - InvalidDataAccessResourceUsageException
  - IncorrectUpdateSemanticsDataAccessException
  - TypeMismatchDataAccessException
  - DataRetrievalFailureException
  - ObjectRetrievalFailureException
  - ObjectOptimisticLockingFailureException
  - OptimisticLockingFailureException
  - DataIntegrityViolationException
  - DeadlockLoserDataAccessException
  - UncategorizedDataAccessException
Reason #4
Aspect-Oriented Programming
What Problem Does AOP Solve?

Aspect-Oriented Programming (AOP) enables modularization of cross-cutting concerns.
What are Cross-Cutting Concerns?

Generic functionality that is needed in many places in your application

- Logging and Tracing
- Transaction Management Security
- Caching
- Error Handling
- Performance Monitoring
- Custom Business Rules
An Example Requirement

- Perform a role-based security check before every application method
An Example Requirement

- Perform a role-based security check before every application method

A sign this requirement is a cross-cutting concern
Without AOP

- Failing to modularize cross-cutting concerns leads to two things
  - Code tangling
  - A coupling of concerns
- Code scattering
  - The same concern spread across modules
Tangling

```java
public class RewardNetworkImpl implements RewardNetwork {
    public RewardConfirmation rewardAccountFor(Dining dining) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
        Account a = accountRepository.findByCreditCard(…
        Restaurant r = restaurantRepository.findByMerchantNumber(…
        MonetaryAmount amt = r.calculateBenefitFor(account, dining);
        …
    }
}
```
public class RewardNetworkImpl implements RewardNetwork {
    public RewardConfirmation rewardAccountFor(Dining dining) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
        Account a = accountRepository.findByCreditCard(...
        Restaurant r = restaurantRepository.findByMerchantNumber(...
        MonetaryAmount amt = r.calculateBenefitFor(account, dining);
        ...
    }
}
public class HibernateAccountManager implements AccountManager {
    public Account getAccountForEditing(Long id) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
    }
    ...
}

public class HibernateMerchantReportingService implements MerchantReportingService {
    public List<DiningSummary> findDinings(String merchantNumber, DateInterval interval) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
    }
    ...
}
public class HibernateAccountManager implements AccountManager {
    public Account getAccountForEditing(Long id) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
    }
    ...
}

public class HibernateMerchantReportingService implements MerchantReportingService {
    public List<DiningSummary> findDinings(String merchantNumber, DateInterval interval) {
        if (!hasPermission(SecurityContext.getPrincipal())) {
            throw new AccessDeniedException();
        }
    }
    ...
}
Without AOP

BankService  CustomerService  ReportingService
Without AOP

Security

BankService

CustomerService

ReportingService

woensdag 6 oktober 2010
Without AOP

Security  Transactions

BankService  CustomerService  ReportingService
Without AOP

- Security
- Transactions
- Logging

BankService
CustomerService
ReportingService
Without AOP

Security
Transactions
Logging

BankService
CustomerService
ReportingService

Code scattering
Without AOP

BankService

CustomerService

ReportingService

Security
Transactions
Logging

Code scattering

Code tangling
How AOP Works

1. Implement your mainline application logic
2. Write aspects to implement your cross-cutting concerns
3. Weave the aspects into your application
AOP based

BankService
CustomerService
ReportingService
AOP based

BankService

CustomerService

Security Aspect

ReportingService
AOP based

BankService

CustomerService

ReportingService

Security Aspect

Transaction Aspect
AOP based

BankService

CustomerService

ReportingService

Security Aspect

Transaction Aspect

Logging Aspect

woensdag 6 oktober 2010
AOP based

BankService

CustomerService

ReportingService

Security Aspect

Transaction Aspect

Logging Aspect
AOP Quick Start

• Consider this basic requirement

Log a message every time a property is about to change

• How can you use AOP to meet it?
public class SimpleCache implements Cache {
    private int cacheSize;
    private DataSource dataSource;

    public void setCacheSize(int size) {
        cacheSize = size;
    }

    public void setDataSource(DataSource ds) {
        dataSource = ds;
    }

    ...
}
The Aspect
The Aspect

@Aspect
The Aspect

@Aspect
public class PropertyChangeTracker {

@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());
The Aspect

```java
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void set*(*))")
```
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void set*(*)")
    public void trackChange() {
        // Code
    }
}
The Aspect

@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void set*(*)")
    public void trackChange() {
        logger.info("Property about to change…");
    }
}
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void set*(\*))")
    public void trackChange() {
        logger.info("Property about to change…");
    }
}
@Aspect
public class PropertyChangeTracker {
    private Logger logger = Logger.getLogger(getClass());

    @Before("execution(void set*(*)")
    public void trackChange() {
        logger.info("Property about to change…");
    }
}
Tell Spring about the Aspect

```xml
<beans>
  
  <aop:aspectj-autoproxy/>

  <bean id="propertyChangeTracker" class="example.PropertyChangeTracker"/>

  <bean name="cache-A" class="example.SimpleCache"/>
  <bean name="cache-B" class="example.SimpleCache"/>
  <bean name="cache-C" class="example.SimpleCache"/>

</beans>
```
Run...

```java
ApplicationContext context =
    new ClassPathXmlApplicationContext("application-config.xml");
Cache cache = (Cache) context.getBean("cache-A");
cache.setCacheSize(2500);
```
Run...

```java
ApplicationContext context =
    new ClassPathXmlApplicationContext("application-config.xml");
Cache cache = (Cache) context.getBean("cache-A");
cache.setCacheSize(2500);
```

INFO: Property about to change…
Reason #5
@Transactional
Why use Transactions?

• Atomic
  – Each unit of work is an all-or-nothing operation

• Consistent
  – Database integrity constraints are never violated

• Isolated
  – Uncommitted changes are not visible to other transactions

• Durable
  – Committed changes are permanent
Local Transaction Management

- Transactions can be managed at the level of a local resource
  - Such as the database
- Requires programmatic management of transactional behavior on the Connection
public void updateBeneficiaries(Account account) {
...
    try {
        conn = dataSource.getConnection();
        conn.setAutoCommit(false);
        ps = conn.prepareStatement(sql);
        for (Beneficiary b : account.getBeneficiaries()) {
            ps.setBigDecimal(1, b.getSavings().asBigDecimal());
            ps.setLong(2, account.getEntityId());
            ps.setString(3, b.getName());
            ps.executeUpdate();
        }
        conn.commit();
    } catch (Exception e) {
        conn.rollback();
        throw new RuntimeException(“Error updating!”, e);
    }
}
Problems with Local Transactions

• Connection management code is error-prone

• Transaction demarcation belongs at the service layer
  – Multiple data access methods may be called within a transaction
  – Connection must be managed at a higher level
public RewardConfirmation rewardAccountFor(Dining dining) {
    Connection conn = DataSourceUtils.getConnection();
    conn.setAutoCommit(false);
    try {
        accountRepository.updateBeneficiaries(account, conn);
        rc = rewardRepository.confirmReward(contrib, dining, conn);
        conn.commit();
    } catch (Exception e) {
        conn.rollback();
        throw new RuntimeException("reward failed", e);
    }
}
Programmatic JTA

- Application Servers enable use of the Java Transaction API (JTA)
- The UserTransaction object is bound to JNDI
- Transactions can be managed from the service layer
- May call multiple data access methods
public RewardConfirmation rewardAccountFor(Dining dining) {
    Context ctx = new InitialContext();
    UserTransaction transaction = (UserTransaction) ctx.lookup("java:comp/UserTransaction");
    transaction.begin();
    try {
        ...
        accountRepository.updateBeneficiaries(account);
        confirmation = rewardRepository.confirmReward(contribution, dining);
        transaction.commit();
    } catch (Exception e) {
        transaction.rollback();
        throw new RuntimeException("failed to reward", e);
    }
}
Programmatic JTA Problems

- Depends on an Application Server environment
- Still requires code to manage transactions
- The code is error-prone
public class RewardNetworkImpl implements RewardNetwork {
    @Transactional
    public RewardConfirmation rewardAccountFor(Dining d) {
        // atomic unit-of-work
    }
}
@Transactional

- Works though AOP
- Consistent approach
  - JDBC
  - JPA
  - Hibernate
  - JMS
  - JTA
  - ...

woensdag 6 oktober 2010
Reason #6
Scripting Languages
Scripting Languages

- More and more popular
- Especially when running on the JVM
- Mix-and-match approach
  - Front-end in JRuby
  - Back-end in Java
Dynamic Language Support in Spring

- Spring container supports
  - Groovy
  - JRuby
  - BeanShell
package org.springframework.scripting;

public interface Messenger {
    String getMessage();
}

require 'java'

class RubyMessenger
  include org.springframework.scripting.Messenger

  def setMessage(message)
    @@message = message
  end

  def getMessage
    @@message
  end
end

<lang:jruby id="messageService"
  script-interfaces="org.springframework.scripting.Messenger"
  script-source="classpath:RubyMessenger.rb">

  <lang:property name="message" value="Hello World!" />

</lang:jruby>
Reason #7

OSGi
JAR Hell

log4j 1.2.13

Spring 2.5.4

Your Application

log4j 1.2.15

Hibernate 3.3.0

woensdag 6 oktober 2010
JAR Hell

log4j 1.2.13

Spring 2.5.4

Hibernate 3.3.0

Your Application

log4j 1.2.15

woensdag 6 oktober 2010
OSGi

log4j 1.2.13

Spring 2.5.4

Your Application

log4j 1.2.15

Hibernate 3.3.0

woensdag 6 oktober 2010
OSGi

log4j 1.2.13

MANIFEST.MF
Import-package:
org.apache.log4j;
version=1.2.13

log4j 1.2.15

MANIFEST.MF
Import-package:
org.apache.log4j;
version=1.2.15

Spring
2.5.4

Hibernate
3.3.0

Your Application

woensdag 6 oktober 2010
OSGi Service contribution

Published services

Private implementation objects
OSGi™ – Dynamic System for Java

- Partition an app into modules
  - Module = Bundle = Jar = a set of classes
- Each module has its own:
  - class space
  - lifecycle
- Strict visibility rules
- Understands versioning
- Everything is *Dynamic*!
OSGi

- Import specific versions of dependencies
- Export as services
Spring Dynamic Modules

- Decorate objects as OSGi services
- Dependency injection for OSGi services
- Remove OSGi dependencies
- Easier to test
Exporting OSGi Service

```xml
<bean id="myPojo" class="someObject"/>
<osgi:service
    ref="myPojo"
    interface="com.springsource.MyService"/>
```
Importing OSGi service

```xml
<osgi:reference
  id="osgiService"
  interface="com.springsource.DynamicService"/>

<bean id="consumer" class..>
  <property name="service" ref="osgiService"/>
</bean>
```
But wait, there is more!

• There is more!
  • JMS, Web, Security, Web Services, Batch, Integration, ...

• Check out www.springframework.org