Aus dem Spring Nähkästchen

Eberhard Wolff
Twitter: @ewolff
Architecture & Technology Manager
adesso AG

https://github.com/ewolff/spring-surprises
https://github.com/ewolff/spring-jms-server
Aus dem Spring Nähhäkchen

Eberhard Wolff
Twitter: @ewolff
Architecture & Technology Manager
adesso AG

https://github.com/ewolff/spring-surprises
https://github.com/ewolff/spring-jms-server
Aus dem Spring Nähk.

Eberhard Wolff
Twitter: @ewolff
Architecture & Technology Manager
adesso AG
Me looking at Spring
Smallest Spring Bean

XML:
Bean = One line of XML

Annotations:
Beans depend on Spring

What is the minimal Spring Bean?
With no code dependencies on Spring?

- Spring != XML
- Convention over configuration possible
- Very versatile and powerful
- Dependency Injection container
- Naming conventions become meaningful
- A way to express your architecture
XML: Bean = One line of XML

What is the minimal Spring Bean?
Annotations: Beans depend on Spring
What is the minimal Spring Bean?
With no code dependencies on Spring?
• Spring != XML
• Convention over configuration possible
• Very versatile and powerful
  Dependency Injection container
• Naming conventions become meaningful
• A way to express your architecture
Smallest Spring Bean

XML:
Bean = One line of XML

Annotations:
Beans depend on Spring

What is the minimal Spring Bean?
With no code dependencies on Spring?

- Spring != XML
- Convention over configuration possible
- Very versatile and powerful
- Dependency Injection container
- Naming conventions become meaningful
- A way to express your architecture
Smallest JMS Server

API abstractions for JMS, JDBC, JCA, JMX, JavaMail, Scheduling ...

Do the abstractions depend on e.g. Spring AOP or DI?

Does it take an Application Server to create a scalable JMS message receiver?

- Great API abstraction
- Spring is a nice JMS, JDBC, JCA, JMX, JavaMail, Scheduling library
- No need to wait for an optimized API
- Uniform concepts (templates, exceptions etc)
- Abstraction can be used w/o XML or Dependency Injection
- Spring’s modularization works for you - no need to use everything!

- Dependency Injection is really a Pattern
- Hand-written DI: Not a great idea
API abstractions for JMS, JDBC, JCA, JMX, JavaMail, Scheduling ...
Do the abstractions depend on e.g. Spring AOP or DI?
Does it take an Application Server to create a scalable JMS message receiver?

- Great API abstraction
- Spring is a nice JMS, JMS, JavaMail, Scheduling library
- No need to reinvent the wheel
scalable JMS receiver?

- Great API abstraction
- Spring is a nice JMS, JDBC, JCA, JMX, JavaMail, Scheduling library
- No need to wait for an optimized API
- Uniform concepts (templates, exceptions etc)
- Abstraction can be used w/o XML or Dependency Injection
- Spring's modularization works for you - no need to use
- No need to wait for an optimizer
- Uniform concepts (templates, exceptions etc)
- Abstraction can be used w/o XML or Dependency Injection
- Spring's modularization works for you - no need to use everything!

- Dependency Injection is really a Pattern
XML or Dependency Injection

- Spring's modularization works for you - no need to use everything!

- Dependency Injection is really a Pattern
- Hand-written DI: Not a great idea
Method Injection

Constructor Injection

Setter Injection

Method Injection?

- Create Domain Objects (?)
- Use differently scoped beans
- No need to call ApplicationContext.getBean()
- DI is universal
- Impress even Spring geeks
Constructor Injection
Setter Injection
Method Injection?
• Create Domain Objects (?)
• Use differently scoped beans
• No need to call ApplicationContext.getBean()
• DI is universal
• Impress even Spring geeks
Fun with @Transactional

Declarative transaction management eliminates important source of mistakes

- Spring AOP / Tx doesn't work on internal method calls
- @Transactional tests: No need to clean up environment
Fun with @Transaction

Declarative transaction management eliminates important source of mistakes
• Spring AOP / Tx doesn't work on internal method calls
• @Transactional tests: No need to clean up environment
Tracing with AOP

How do you implement it?

- Look at org.springframework.aop.interceptor and org.springframework.util for some nice helpers
Aspekt-Orientiertes Programmieren

- Wie man diese Art von Rat nutzt: währenddessen, davor, danach
racing with AOP

How do you implement it?

- execution(* hello())
  - Execution of method hello
  - any parameters
  - any return type

- execution(* hello(..))
  - Execution of method hello
  - any parameters
  - any return type

- execution(* package.Class.*(..))
  - Execution of any method in package.Class
  - any parameters
  - any return type

- execution(* ..*Repository)
  - Execution of any method in package and all subpackages
  - Class name ends in Repository

- execution(* logging..*..*(..))
  - Execution of any method
Some Pointcuts
execution(void hello())
Execution of method hello
no parameters
void return type
execution(* hello())
Execution of method hello
no parameters
any return type
execution(* hello(..))
Execution of method hello
any parameters
any return type
execution(* package.Class.*(..))

Execution of any method in package.Class

any parameters

any return type
execution(* *..*Repository.*(..))
Execution of any method in any package and all subpackages
Class name ends in Repository

execution(* logging..*.*(..))
Execution of any method in
execution(* logging..*.*(..))
Execution of any method in package logging and all subpackages
Any class name
• Look at `org.springframework.aop.interceptor` and `org.springframework.util` for some nice helpers
Aus dem Spring Nähkästchen

Eberhard Wolff
Twitter: @ewolff
Architecture & Technology Manager
adesso AG

https://github.com/ewolff/spring-surprises
https://github.com/ewolff/spring-jms-server
Smallest Spring Bean

XML:
Bean = One line of XML

Annotations:
Beans depend on Spring

What is the minimal Spring Bean?
With no code dependencies on Spring?

- Spring ≠ XML
- Convention over configuration possible
- Very versatile and powerful
- Dependency Injection container
- Naming conventions become meaningful
- A way to express your architecture
Smallest JMS Server

API abstractions for JMS, JDBC, JCA, JMX, JavaMail, Scheduling ...

Do the abstractions depend on e.g. Spring AOP or DI?

Does it take an Application Server to create a scalable JMS message receiver?

- Great API abstraction
- Spring is a nice JMS, JDBC, JCA, JMX, javaMail, Scheduling library
- No need to wait for an optimized API
- Uniform concepts (templates, exceptions etc)
- Abstraction can be used w/o XML or Dependency Injection
- Spring’s modularization works for you - no need to use everything!
- Dependency Injection is really a Pattern
- Hand-written DI: Not a great idea
Method Injection

**Constructor Injection**

**Setter Injection**

**Method Injection?**

- Create Domain Objects (?)
- Use differently scoped beans
- No need to call `ApplicationContext.getBean()`
- DI is universal
- Impress even Spring geeks
Fun with @Transactional

Declarative transaction management eliminates important source of mistakes

- Spring AOP / Tx doesn't work on internal method calls
- @Transactional tests: No need to clean up environment
Tracing with AOP

How do you implement it?

- Look at `org.springframework.aop.interceptor` and `org.springframework.util` for some nice helpers
Aus dem Spring Nähkästchen

Eberhard Wolff
Twitter: @ewolff
Architecture & Technology Manager
adesso AG

https://github.com/ewolff/spring-surprises
https://github.com/ewolff/spring-jms-server