Spring Framework Essentials

Ken Kousen
ken.kousen@kousenit.com

Java EE:
- components
  - jar, war, ear
- containers
  - web, ejb
- services
  - security
  - transactional
  - persistence
  - resource pooling
  - ..

meta-data
- XML
  - deployment descriptor

Lightweight container
- provide services to POJOs

Spring 1.0 → 2004
Spring 2.0 → 2006
- simplified XML config

Spring 2.5 → 2007
- annotation configuration

Spring 3.0 → 2010 to 2012
- Spring 3.2 → 2012
- Support for Java 7, Hibernate 4, Servlet 3.0

- Required Java 1.5+
- Java configuration approach

Spring 4.0 → 2014

- Supports Java 8

Spring Boot → 2014

Spring favors interface/class separation

- Write your code in terms of interfaces
- Tell Spring which classes to provide
- "wire everything together"

Inversion of Control

- container (IoC)

Injecting Dependencies

A dependency is simply an attribute

- method argument
- return type

Injection → calling a setter method, or using a constructor argument

Metadata describes the actual classes and services that we want
Spring wires everything together and manages the lifecycle

We rarely instantiate beans
We don't look them up

```java
package com.oreilly;

import com.oreilly.entities.*;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AppConfig {
   @Bean
   public Game game() {
       return new BaseballGame(redSox(), cubs());
   }

   @Bean
   public Team redSox() {
       return new RedSox();
   }

   @Bean
   public Team cubs() {
       return new Cubs();
   }
}
```

@Autowired means "autowire by type" first

    This works if there is exactly one bean of that type (class) available

@Resource → Java standard annotation

    This uses "autowire by name"

What often happens, is you make a package for each type you want to be found on a component scan

    repositories
    services
    controllers

By default, all Spring managed beans are singletons!

    @Scope("singleton")

other options are "prototype"
if you are in a web app (Spring MVC),
"request", "session"

```java
@Bean
public Team redSox() {
   return new RedSox();
}
```

In subclass:

```
public Team redSox() {
        if (redSox already in appContext) {
                return redSox;
        } else {
                // call super.redSox()
                // add it to the appContext
                // return redSox
        }
}
```

```java
@Bean(initMethod = "startUp", destroyMethod = "cleanUp")
```

In bean,

```java
        @PostConstruct
        public void startUp()

        @PreDestroy
        public void cleanUp()
```

AOP:

Code tangling
Code scattering

Join Point

public methods in Spring-managed beans

Pointcut

The actual joinpoints that we have declared

Advice

The functionality we want to apply

Aspect

Combines pointcut and advice

Weaving

The process of applying an aspect to our system

Advice types:

- Before
- After
- AfterReturning
- AfterThrowing
- Around

Transactions

- ACID Properties →
- Atomic
    - All or nothing
- Consistent
    - DB integrity constraints never violated
- Isolated
    - How transactions see work done by others
- Durable
    - Committed changes are permanent

Transactions in Spring

1. Apply the @Transactional annotation
    a. XML format
    b. Programmatically
2. Declare a Platform Transaction Manager bean
3. @EnableTransactionManagement

Propagation Levels
- REQUIRED (default)
    - tx → join tx
    - no → create and run tx
- REQUIRES_NEW
    - tx1 → suspend tx1; create and run tx2; resume tx1
    - no → create and run tx
- SUPPORTS
    - tx → join tx
    - no → nothing
- NOT_SUPPORTED
    - tx → suspend tx; run outside tx; resume tx
    - no → nothing
- MANDATORY
    - tx → join tx
    - no → throw a TransactionRequiredException
- NEVER
    - tx → throw an exception
    - no → nothing

JPA → the Java Persistence API
- Extracted from EJB 3
- JPA 2, Dec 2009; JPA 2.1, May 2013

EntityManagerFactory
EntityManager
persistence.xml