Spring Framework

Intro

About Spring

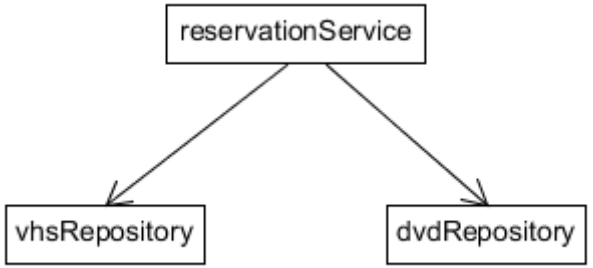
- Create in 2002 by Rod Johnson (Interface21)
  - Expert One-on-One J2EE Design and Development
- Current version 4.3 (ongoing)
- Open Source, Apache License
  - Free sources, for commercial use, even derivates
- Perfect documentation (HTML + javadoc)
- High quality API + extensions
- Owned and supported by VMWare

Introduction

- What is Spring (container + modules)
- Library – JAR file(s)
  - Standalone + helpers components
- Container
  - Component lifecycle
    - Instantiate
    - Wire them to together (Dependency Injection)
    - Supply with services (transation, exception handling...)
- Spring Tool Suite

Spring Container - definition

https://github.com/zbynekvavros/spring-intro.git




```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="vhsRepository"
        class="net.test.persistence.repository.VhsRepositoryImpl" />

    <bean id="dvdRepository"
        class="net.test.persistence.repository.DvdRepositoryImpl" />

    <bean id="reservationService"
        class="net.test.services.ReservationServiceImpl">
        <property name="vhsRepository" ref="vhsRepository" />
        <property name="dvdRepository" ref="dvdRepository" />
    </bean>

</beans>
```

Spring Container - instantiate

- Instantiate Spring components

```
ApplicationContext applicationContext = new ClassPathXmlApplicationContext("applicationContext.xml");
```

  - Spring instantiates all components

- Look-up the components

```
ReservationService reservationService = applicationContext.getBean(ReservationService.class, "reservationService");
```

Dependencies

- Property → setter
- Constructor
  - Mandatory vs optional
- Other values
  - NULL
  - Primitives
  - Collections
  - Properties
  - etc

Terminology

- Component
  - Every <bean>
  - No need for JavaBeans only (contructor)
  - Ideally with interface
- Inversion of Control
  - Design pattern
  - Difference between main() and Servlet
- Dependency injection
  - Components doesn't perform the injection

Advantages of DI

- No instantiation (new operator)
- Eliminates lookup (Service Locator)
- Prohibits cyclic references
- Help and encourages unit testing
- Ideally with interfaces
  - Easier implementation replacement
  - Forces developer to better code design

Additional lifecycle management

- Initialization
  - InitializingBean, init-method, @PostConstruct
- Closing
  - DisposableBean, destroy-method, @PreDestroy
- Scopes
  - Singleton, Prototype, Session, Custom

Component scan + Autowiring

- No need to configure each bean in XML
- @Component, @Controller, @Service, @Repository
- No need for <property> or <constructor-arg>
- @Autowired

Java Config

- Completely without XML
- Allows more customization
- May need time to get used to

Properties files

- Declaration XML

```
<context:property-placeholder location="classpath:properties/dummy.properties" />
```

- Declaration annotation

```
@PropertySource("classpath:properties/dummy.properties")

@Bean
public static PropertyPlaceholderConfigurer propertyPlaceholderConfigurer() {
    return new PropertyPlaceholderConfigurer();
}
```

- Usage

```
@Autowired
private Environment environment;


environment.getProperty("properties.int");
```

```
@Value("${properties.int}")
private Long foo;
```

Pros / Cons

- Scaling, large API, documentation, integration
- Uses and forces right design pattern
- No magic :)
- No need to change code
-
- Not a JSR standard
- Evolution can be confusing

Exercise ?

https://github.com/zbynekvavros/spring-intro-exercise.git