Introduction to Spring. The Best Java framework.

IT Learning &
Outsourcing Center

www.pragmatic.bg

Lector: Stefan Vadev
Skype: stefanvadev77
E-mail: stefan.vadev@gmail.com
Facebook:
https://www.facebook.com/stefan.vadev.7

Summary

- Spring overview
- Spring architecture
- Inversion of control
- Dependency injection
- Spring configuration
- Spring beans
- More

Spring

Spring overview

Spring framework is an open source Java platform that provides comprehensive infrastructure support for developing robust Java applications very easily and very rapidly. Spring framework was initially written by Rod Johnson and was first released under the Apache 2.0 license in June 2003.

Spring is the most popular application development framework for enterprise Java. Millions of developers around the world use Spring Framework to create high performing, easily testable, and reusable code.

Spring framework is an open source Java platform.

Spring overview

Spring is lightweight when it comes to size and transparency. The basic version of Spring framework is around 2MB.

The core features of the Spring Framework can be used in developing any Java application, but there are extensions for building web applications on top of the Java EE platform. Spring framework targets to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.

Benefits of Spring

Spring enables developers to develop enterprise-class applications using POJOs. The benefit of using only POJOs is that you do not need an EJB container product such as an application server but you have the option of using only a robust servlet container such as Tomcat or some commercial product.

Spring is organized in a modular fashion. Even though the number of packages and classes are substantial, you have to worry only about the ones you need and ignore the rest.

Benefits of Spring

Spring does not reinvent the wheel, instead it truly makes use of some of the existing technologies like several ORM frameworks, logging frameworks, JEE, Quartz and JDK timers, and other view technologies.

Testing an application written with Spring is simple because environment-dependent code is moved into this framework. Furthermore, by using Java Bean style POJOs, it becomes easier to use dependency injection for injecting test data.

Benefits of Spring

Spring's web framework is a well-designed web MVC framework, which provides a great alternative to web frameworks such as Struts or other over-engineered or less popular web frameworks.

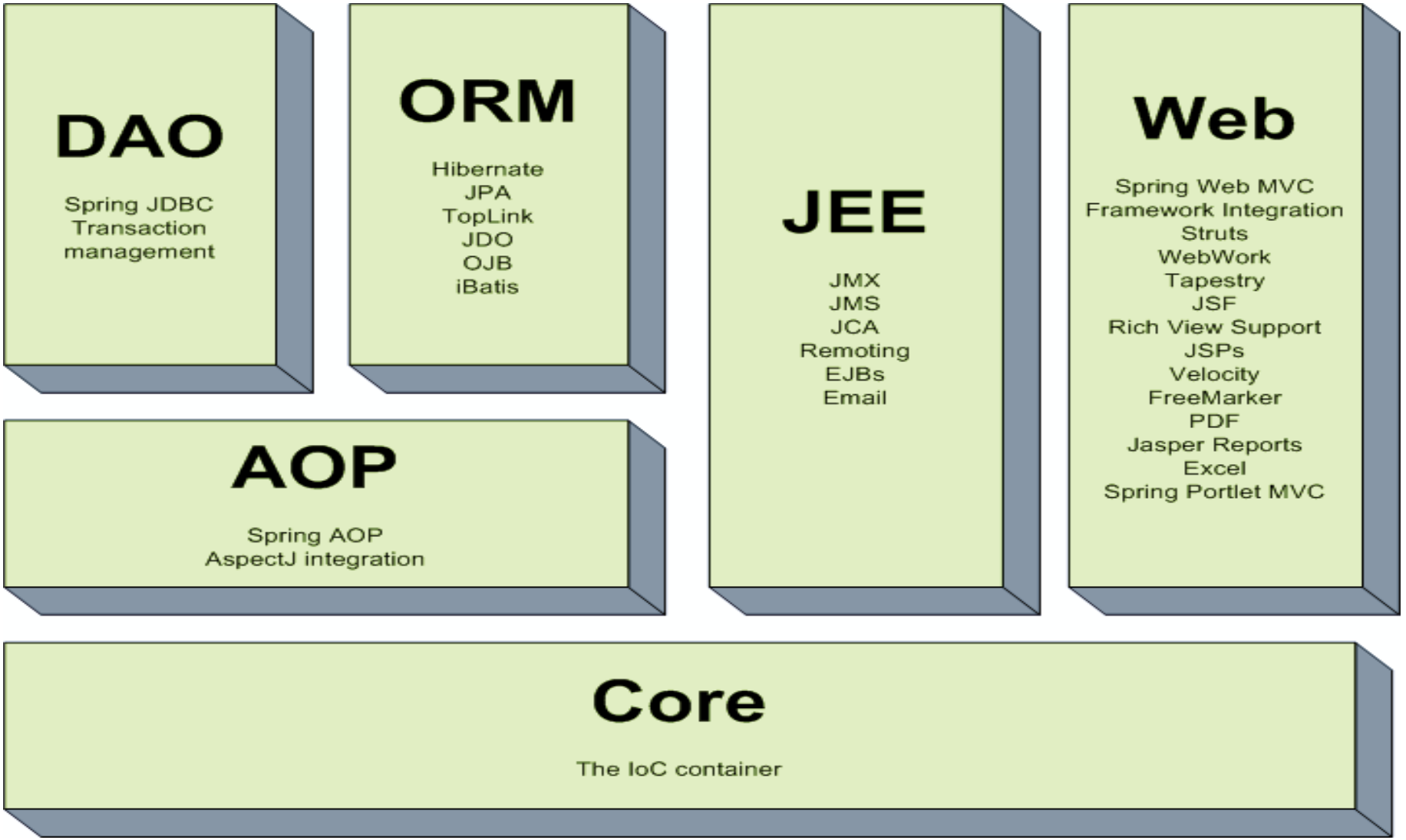
Spring provides a convenient API to translate technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

Benefits of Spring

Lightweight IoC containers tend to be lightweight, especially when compared to EJB containers, for example. This is beneficial for developing and deploying applications on computers with limited memory and CPU resources.

Spring provides a consistent transaction management interface that can scale down to a local transaction (using a single database, for example) and scale up to global transactions (using JTA, for example).

Spring architecture



Dependency injection

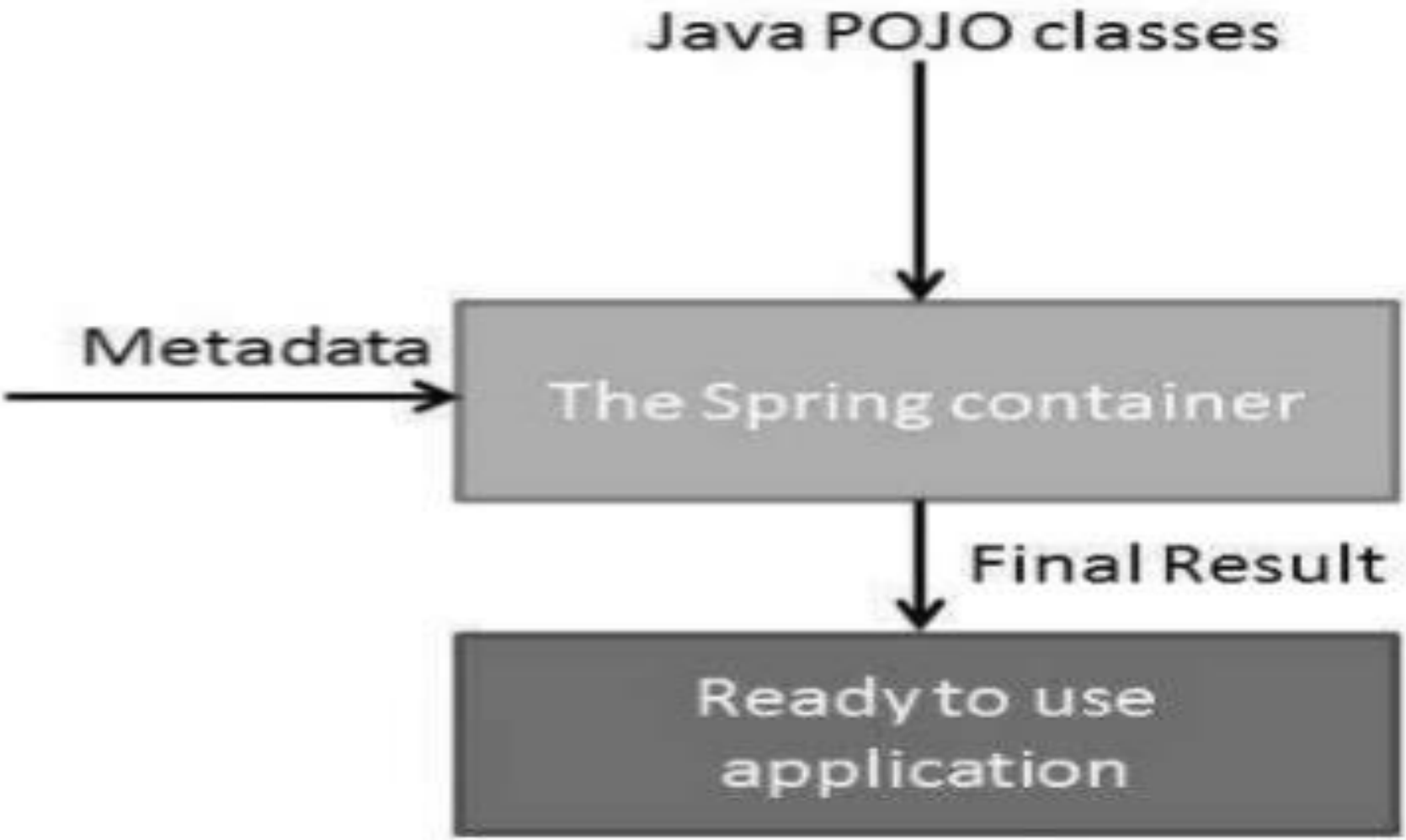
The technology that Spring is most identified with is the Dependency Injection (DI) flavor of Inversion of Control. The Inversion of Control (IoC) is a general concept, and it can be expressed in many different ways. Dependency Injection is merely one concrete example of Inversion of Control.

When writing a complex Java application, application classes should be as independent as possible of other Java classes to increase the possibility to reuse these classes and to test them independently of other classes while unit testing. Dependency Injection helps in gluing these classes together and at the same time keeping them independent.

Dependency injection

Dependency injection can happen in the way of passing parameters to the constructor or by post-construction using setter methods. As Dependency Injection is the heart of Spring Framework.

Inversion of control



IOC – Inversion of control

The Spring container is at the core of the Spring Framework. The container will create the objects, wire them together, configure them, and manage their complete life cycle from creation till destruction. The Spring container uses DI to manage the components that make up an application. These objects are called Spring Beans.

IOC – Inversion of control

The container gets its instructions on what objects to instantiate, configure, and assemble by reading the configuration metadata provided. The configuration metadata can be represented either by XML, Java annotations, or Java code. The following diagram represents a high-level view of how Spring works. The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.

Application context

This container adds more enterprise-specific functionality such as the ability to resolve textual messages from a properties file and the ability to publish application events to interested event listeners. This container is defined by the ***org.springframework.context.ApplicationContext interface.***

Aspect Oriented programming

One of the key components of Spring is the Aspect Oriented Programming (AOP) framework. The functions that span multiple points of an application are called cross-cutting concerns and these cross-cutting concerns are conceptually separate from the application's business logic. There are various common good examples of aspects including logging, declarative transactions, security, caching, etc.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. DI helps you decouple your application objects from each other, while AOP helps you decouple cross-cutting concerns from the objects that they affect.

Spring Documentation

Spring configuration and more

https://docs.spring.io/spring-framework/docs/current/Javadoc-api/org/springframework/context/annotation/Configuration.html

Use documentation whenever needed, it is very good and very useful:

https://spring.io/docs

https://spring.io/docs/reference

For example: Spring boot docs: https://docs.spring.io/spring-boot/docs/2.0.1.BUILD-SNAPSHOT/reference/htmlsingle/

Spring beans

The objects that form the backbone of your application and that are managed by the **Spring IoC** container are called **beans**. A bean is an object that is instantiated, assembled, and otherwise managed by a Spring IoC container. These beans are created with the configuration metadata that you supply to the container.

Spring beans

https://docs.spring.io/spring-javaconfig/docs/1.0.0.M4/reference/html/ch02s02.html

Scope of beans

Bean Scopes

Scope	Description
Singleton	(Default)Only one single instance will be created
Prototype	Creates any number of instances from a single bean configuration
Request	Scope of the bean instance will be limited to the Request life cycle
Session	Limited to session
Global session	Limited to global session-Portlet context.

<bean name ="student" class ="Student" **scope ="prototype"**/>

Spring configuration types

XML configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans" ...>

   <bean id="accountService" class="com.wiley.beginningspring.ch2.AccountServiceImpl">
      <property name="accountDao" ref="accountDao"/>
   </bean>

   <bean id="accountDao"
class="com.wiley.beginningspring.ch2.AccountDaoInMemoryImpl">
   </bean>

</beans>
```

```
ClassPathXmlApplicationContext applicationContext = new
ClassPathXmlApplicationContext("/com/wiley/beginningspring/ch2/ch2-beans.xml");
```

Spring configuration types

Java based configuration

```
@Configuration
public class Ch2BeanConfiguration {

    @Bean
    public AccountService accountService() {
        AccountServiceImpl bean = new AccountServiceImpl();
        bean.setAccountDao(accountDao());
        return bean;
    }

    @Bean
    public AccountDao accountDao() {
        AccountDaoInMemoryImpl bean = new AccountDaoInMemoryImpl();
        return bean;
    }
}
```

Spring configuration types

Annotation based configuration

```
@RestController
@RequestMapping("/dateconverter")
public class DateConverterController {

    private final DateConverterService service;

    @Autowired
    public DateConverterController(DateConverterService service) {
        this.service = service;
    }

    @RequestMapping("")
    public String convert(@RequestParam(value = "date") String date) {
        return service.convert(date);
    }
}
```

Core Spring Framework Annotations

@Autowired

This annotation is applied on fields, setter methods, and constructors. The @Autowired annotation injects object dependency implicitly.

When you use @Autowired on fields and pass the values for the fields using the property name, Spring will automatically assign the fields with the passed values.

@Autowired is used on constructor level as well (best use!).

You can even use @Autowired on private properties, as shown below. (This is a very poor practice though!)

```
public class Customer {
    @Autowired
    private Person person;
    private int type;
}
```

Spring annotations

@Required

This annotation is applied on bean setter methods. Consider a scenario where you need to enforce a required property. The @Required annotation indicates that the affected bean must be populated at configuration time with the required property. Otherwise an exception of type **BeanInitializationException** is thrown.

Spring annotations

@Qualifier

This annotation is used along with @Autowired annotation. When you need more control of the dependency injection process, @Qualifier can be used. @Qualifier can be specified on individual constructor arguments or method parameters. This annotation is used to avoid confusion which occurs when you create more than one bean of the same type and want to wire only one of them with a property.

Spring annotations

@Configuration

This annotation is used on classes which define beans. @Configuration is an analog for XML configuration file – it is configuration using Java class. Java class annotated with @Configuration is a configuration by itself and will have methods to instantiate and configure the dependencies. Here is an example:

```
@Configuration
public class DataConfig{
  @Bean
  public DataSource source(){
    DataSource source = new OracleDataSource();
    source.setURL();
    source.setUser();
    return source;
  }
  @Bean
  public PlatformTransactionManager manager(){
    PlatformTransactionManager manager = new BasicDataSourceTransactionManager();
    manager.setDataSource(source());
    return manager;
  }
}
```

Spring annotations

@ComponentScan

This annotation is used with **@Configuration** annotation to allow Spring to know the packages to scan for annotated components. **@ComponentScan** is also used to specify base packages using **basePackageClasses** or **basePackage** attributes to scan. If specific packages are not defined, scanning will occur from the package of the class that declares this annotation.

Spring annotations

@Bean

This annotation is used at the method level. **@Bean** annotation works with **@Configuration** to create Spring beans. As mentioned earlier, **@Configuration** will have methods to instantiate and configure dependencies. Such methods will be annotated with **@Bean**. The method annotated with this annotation works as bean ID and it creates and returns the actual bean.

Here is an example:

```
@Configuration
public class AppConfig{
  @Bean
  public Person person(){
    return new Person(address());
  }
  @Bean
  public Address address(){
    return new Address();
  }
}
```

Spring annotations

@Lazy

This annotation is used on component classes. By default all autowired dependencies are created and configured at startup. But if you want to initialize a bean lazily, you can use @**Lazy** annotation over the class. This means that the bean will be created and initialized only when it is first requested for. You can also use this annotation on @**Configuration** classes. This indicates that all @**Bean** methods within that @**Configuration** should be lazily initialized.

Spring annotations

@Value

This annotation is used at the field, constructor parameter, and method parameter level. The **@Value** annotation indicates a default value expression for the field or parameter to initialize the property with. As the **@Autowired** annotation tells Spring to inject object into another when it loads your application context, you can also use **@Value** annotation to inject values from a property file into a bean's attribute. It supports both #{...} and ${...} placeholders.

Spring Framework Stereotype Annotations

@Component

This annotation is used on classes to indicate a Spring component.
The **@Component** annotation marks the Java class as a bean or say component so that the component-scanning mechanism of Spring can add into the application context.

Spring Framework Stereotype Annotations

@Controller

The @Controller annotation is used to indicate the class is a Spring controller. This annotation can be used to identify controllers for Spring MVC or Spring WebFlux.

Spring Framework Stereotype Annotations

@Service

This annotation is used on a class. The @Service marks a Java class that performs some service, such as execute business logic, perform calculations and call external APIs. This annotation is a specialized form of the@Component annotation intended to be used in the service layer.

Spring Framework Stereotype Annotations

@Repository

This annotation is used on Java classes which directly access the database. The **@Repository** annotation works as marker for any class that fulfills the role of repository or Data Access Object.

This annotation has a automatic translation feature. For example, when an exception occurs in the **@Repository** is a handler for that exception and there is no need to add a try catch block.

Spring annotations

More info about Spring annotations here:

https://springframework.guru/spring-framework-annotations/

Summary

- Spring overview
- Spring architecture
- Inversion of control
- Dependency injection
- Spring configuration
- Spring beans
- More