Spring Framework and Dependency Injection

Ahmet Arif Aydin
Agenda

- What is Spring Framework
- Intro to Spring
- What are Beans?
- Big Picture of Spring
- Two Key Components of Spring (AOP & DI)
- Spring Framework Architecture
  - Core Container Modules
  - Data Access/Integration Layer Modules
  - Web Layer Modules
- Dependency Injection (DI) Types
- Constructor-based Dependency Injection
- Setter-based Dependency Injection
What is Spring Framework?

• Spring Framework was created by Rod Johnson (2003) and released under Apache 2.0 license.
• The most popular application development framework for enterprise Java
• An Open source Java platform
• Provides to create high performing, easily testable and reusable code.
• is organized in a modular fashion
• simplifies java development
Spring Framework

- enables Plain Old Java Object (POJO) based programming model
- with POJO you don’t need EJB container product
- utilizes existing technologies like
  - ORM frameworks
  - logging frameworks
  - JEE
  - Quartz
  - JDK timers
Spring Framework

- is a well-designed web model-view-controller (MVC) framework (a great alternative to Struts)
- provides a coherent transaction management interface that be applicable to a local transactions() local transactions or global transactions(JTA)
- provides a suitable API for translating technology-specific exceptions (for instance, thrown by JDBC, Hibernate, or JDO,) into consistent, unchecked exceptions.
- The Inversion of Control (IoC) containers are lightweight, especially when compared to EJB containers. Being lightweight is beneficial for developing and deploying applications on computers with limited resources (RAM&CPU).
- Testing is simple because environment-dependent code is moved into this framework.
What are Beans?

- In Spring, POJO’s (plain old java object) are called ‘beans’ and those objects instantiated, managed, created by Spring IoC container.
- Beans are created with the configuration metadata (XML file) that we supply to the container.
- Bean definition contains configuration metadata. With this information container knows how to create bean, beans lifecycle, beans dependencies
- After specifying objects of an application, instances of those objects will be reached by getBean() method.
- Spring supports given scope types for beans:
  - Singleton (a single instance per Spring IoC container (default))
  - Prototype
  - Request
  - Session
  - Global-session
Scope of Beans

- Spring supports given scope types for beans:
  - Singleton (a single instance per Spring IoC container (default))
  - Prototype
  - Request
  - Session
  - Global-session

<!-- A bean definition with singleton scope -->
<bean id="..." class="..." scope="singleton">
  <!-- collaborators and configuration for this bean go here -->
</bean>
The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.
Two Key Components of Spring

- **Dependency Injection (DI)** helps you decouple your *application objects* from each other.
- **Aspect Oriented Programming (AOP)**
  - The key unit of modularity is the *aspect* in AOP (class in OOP)
  - **Cross-cutting concerns** are the functions that span multiple points of an application.
  - Cross-cutting concerns are conceptually separate from the application's business logic.
  - AOP helps you decouple *cross-cutting concerns from the objects* that they affect. Examples (logging, declarative transactions, security, and caching).
Spring Framework Architecture

- **Core Module**: The Spring container is at the core module.
  - The Spring container is responsible to create objects, wire them together and manage them from creation until destruction.
  - The Spring container utilizes Dependency Injection to manage objects that make up an application.
Spring Framework Architecture

- **Beans Module** provides BeanFactory, preferred when the resources are limited such as mobile devices or applet based applications.
Spring Framework Architecture

- **Context Module** builds on the solid base provided by the Core and Beans modules and it (medium to access any objects defined and configured)

- ApplicationContext Container (Spring’s more advanced container). This includes all functionality of BeanFactory. The most commonly used implementations are:
  - **FileSystemXmlApplicationContext** (loads definitions of the beans from an XML file. Need to provide full path of xml file)
  - **ClassPathXmlApplicationContext** loads definitions of the beans from an XML file. Does not need to provide the full path it will work with the xml file in the Classpath)
  - **WebXmlApplicationContext** (loads the XML file with definitions of all beans from within a web application.)
Spring Framework Architecture

- The JDBC (provides a JDBC-abstraction layer that removes the need to JDBC related coding)
- The ORM (provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis)
- The OXM provides an abstraction layer that supports Object/XML mapping implementations for JAXB, Castor, XMLBeans, JiBX and XStream.
- The Java Messaging Service (features for producing and consuming messages.)
- The Transaction module supports programmatic and declarative transaction management for classes that implement special interfaces and for all your POJOs.
Spring Framework Architecture

- The Web module provides
  - Basic web-oriented integration features (e.g., multipart file-upload functionality and the initialization of the IoC container using servlet listeners and a web-oriented application context).

- The Web-Servlet module contains Spring's MVC implementation for web applications.

- The Web-Struts module contains the support classes for integrating a classic Struts web tier within a Spring application.

- The Web-Portlet module provides the MVC implementation to be used in a portlet environment and mirrors the functionality of Web-Servlet module.
Dependency Injection (DI)

- Spring is most identified with **Dependency Injection (DI)** technology.
- DI is only one concrete example of Inversion of Control.
- In a complex Java application, classes should be loosely coupled. This feature provides code reuse and independently testing classes.
- DI helps in gluing loosely coupled classes together and at the same time keeping them independent.
- Using dependency injection helps to see easily what the component dependencies are.
- DI is preferable because it makes testing easier.
Dependency Injection Types

- DI will be accomplished by given two ways:
  - *passing parameters* to the constructor (*used for mandatory dependencies*) or
  - using *setter methods* (*used for optional dependencies*).
Constructor-based DI

- Constructor based DI occurs when the container invokes a class constructor with a number of arguments, each representing a dependency on other class.
Constructor-based DI (Plane.java)

```java
public class Plane {
    private RouteFinder routeChecker;

    public Plane (RouteFinder routeChecker) {
        System.out.println("Inside Plane Constructor");
        this.routeChecker = routeChecker;
    }

    public void routeCheck() {
        routeChecker.findRoute();
    }
}
```
public class RouteFinder {
    public RouteFinder() {
        System.out.println("Inside RouteFinder's constructor");
    }
    public void findRoute() {
        System.out.println("Inside findRoute method in RouteFinder ");
    }
}
Constructor-based DI (RouteTest.java)

```java
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RouteTest {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");
        Plane rc = (Plane) context.getBean("plane");
        rc.routeCheck();
    }
}
```

INFO: Pre-instantiating singletons in org.springframework
Inside RouteFinder's constructor
Inside Plane Constructor
Inside findRoute method in RouteFinder
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    <bean id="plane" class="Plane">
        <constructor-arg ref="route"/>
    </bean>

    <bean id="route" class="RouteFinder"/>

</beans>
public class Plane {
    private RouteFinder routeChecker;

    // a setter method to inject the dependency.
    public void setRoute(RouteFinder routeChecker) {
        System.out.println("Inside setRoute method in Plane");
        this.routeChecker = routeChecker;
    }

    // a getter method to return routeChecker
    public RouteFinder getRoute() {
        return routeChecker;
    }

    public void routeCheck() {
        routeChecker.findRoute();
    }
}
public class RouteFinder {

    public RouteFinder() {
        System.out.println("Inside RouteFinder's constructor");
    }

    public void findRoute() {
        System.out.println("Inside findRoute method in RouteFinder ");
    }

}
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class RouteTest {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        Plane rc = (Plane) context.getBean("plane");

        rc.routeCheck();

    }

}
<bean id="plane" class="Plane">
   <property name="route" ref="route"/>
</bean>

<bean id="route" class="RouteFinder" />

</beans>
Conclusion

- The most popular application development framework for enterprise Java
- Spring Framework (Architecture) is modular and allows you to pick and choose modules that are applicable to your application.
- POJO’s (plain old java object) are called ‘beans’ and those objects instantiated, managed, created by Spring IoC container.
- The Spring IoC container makes use of Java POJO classes and configuration metadata to produce a fully configured and executable system or application.
- DI helps in gluing loosely coupled classes together and at the same time keeping them independent.
References

- Spring Framework 3.1 Tutorial
Questions/Discussions