Enterprise Programmering 1

Lesson 08: Spring

Prof. Andrea Arcuri

About these slides

- These slides are just high level overviews of the topics covered in class
- The details are directly in the code comments on the Git repository

Spring Framework

- Open-source, first released in 2002
- Framework to develop enterprise/web applications
- Supported/developed by *Pivotal Software*
- Started as a lightweight alternative to JEE
- Now quite complex, but shares/reuses many aspects of JEE
- Probably one of the frameworks/libraries with the best documentation out there
  - eg, see https://spring.io

Why the name “Spring”?

Spring is what comes after the “*winter*” of JEE…

Spring vs SpringBoot

- Spring has *many*, *many* modules, like for handling databases, web pages, web services, etc.
- To *wire* together a Spring application, there might be the need to set up a lot of configurations
- *SpringBoot* (2012): part of Spring
- Provides *convention over configuration*
  - ie, default, reasonable configurations
  - eg, if you have H2 embedded database as dependency in *pom.xml*, SpringBoot will automatically start it and configure Hibernate for it
- Can write up a full functional enterprise application very quickly

Popularity

- Spring is the most popular framework to develop backend enterprise applications
- However, like JEE, it has a learning curve, as you need to have a clear understanding of *dependency injection* and *proxy classes*
- Java also has other more lightweight alternatives, like *DropWizard*
  - less “magic”, but more boilerplate and less functionalities…

Deployment

- With Spring, you can still build a WAR file that can be deployed on a JEE container like WildFly
- That because Spring still implements the “Servlet” API of JEE (which is the one used to interact with HTTP requests)
- However, most common case is to build a fat / uber JAR file, which is self executable (and contains all needed dependencies)
  - Which will include an embedded servlet container like Tomcat or Jetty
  - These two are servers that support the “Servlet” API, but not the other specs of JEE

From JEE to Spring

- Spring is inspired by JEE, and follows many of its conventions
  - and actually several aspects in JEE came from Spring…
- Spring is huge, so we will just discuss what we have seen so far in JEE, ie: *JPA*, *EJB* and *JSF*
- As they have many similarities, moving from one to the other is not a too complex task

JPA

- Spring can use JPA directly
- By default, Spring uses Hibernate for JPA
- No need to do any change on the *@Entity* or the *EntityManager*
- No need of *persistence.xml*, as Hibernate gets configured in the main Spring configuration file (e.g., *application.yml*)
- Spring provides *more functionalities* on top of JPA, like *@Repository*

EJB

- Not supported in Spring
- Spring has its own equivalent bean mechanism
- Classes annotated with *@Service*, and injection done with *@Autowired*
- Spring services are by default like a *singleton*
- If you want to have transactions, not on by default, you need to use the annotation *@Transactional*
- Often, transforming a EJB into a Spring Service is just a matter of changing just the annotations…

JSF

- Not directly supported by Spring
- Spring has its own server-side HTML rendering framework called *Spring MVC*
- However, JSF (the controller beans and XHTML templates) can be used in Spring with the help of external libraries (e.g., *JoinFaces*)
  - eg, it provides the ability to use Spring beans inside JSF controller beans, and setups Spring to use the JSF Servlet to handle incoming HTTP requests
- Recall: today, if you really want to invest time in learning a frontend framework/library, learn *React.* We use JSF because can be in both JEE and Spring, and want to show how to build a web app without JavaScript
  - ie, no AJAX nor WebSockets

Testing

- This is where Spring completely *annihilates* JEE
- No need of special Maven plugins to download a JEE container like WildFly
  - Spring is just yet another JAR library imported as a dependency
- A Spring application can be started directly by the tests inside the JVM of the tests
  - No need to use Arquillian to create a WAR file on the fly, and then have to control an external process running the JEE container

Entry Point

- You need a class annotated with *@SpringBootApplication*
  - It will be the entry point of your application
- SpringBoot will automatically scan all the classes on your *classpath* (ie your own classes and all third-party dependencies) to check which *@Service* beans to start
- Be CAREFUL of package names: by default, if *@SpringBootApplication* is in package X.Y.Z, it will scan only X.Y.Z and sub-packages
  - ie, X.Y.Z.W is OK, whereas X.Y will be ignored

Configurations

- SpringBoot provides sensible default configurations based on what present on your *classpath*
- If you need to do modifications, those will be in a *application.properties* or *application.yml* file
  - those are the same, just in different formats
  - ".properties": pairs <property-name>=<value>
  - ".yml"/".yaml": YAML (YAML Ain't Markup Language)
- See following for list of properties:
  - https://docs.spring.io/spring-boot/docs/current/reference/html/common-application-properties.html

Git Repository Modules

- *NOTE: most of the explanations will be directly in the code as comments, and not here in the slides*
- **intro/spring/bean/service**
- **intro/spring/bean/jpa**
- **intro/spring/bean/profile**
- **intro/spring/bean/configuration**
- **intro/spring/jsf**
- Exercises for Lesson 08 (see documentation)