Spring

Spring:

* + Is an application framework and inversion of control container for java.
  + Associated with developing websites, very secure, backend, databases
  + Java framework that makes DI easy for developers.
  + Create and store dependencies in the container and injects them into registered objects (beans).

Spring Bean: Represents Java Objects, managed by IOC container, configured in beans.xml file or annotations. For pojos to be configured as spring beans, always have public getters and setters and no args constructor.

String Bean container:

* + Reads beans.xml and instantiates the bean object

ApplicationContext context = new ClassPathXmlApplicationContext(“beans.xml”).

Web container: XmlWebApplicationContext

DI (Dependency Injection)

* + An implementation or “flavor of” IoC.
  + Dependency is some object to be used.
  + Injection is passing dependency to dependent object at creation time.
  + Decouples task execution from implementation

IoC (Inversion of Control): A design pattern in which custom-written portions of a program receive the flow of control from a framework, reduce spaghetti dependencies.

Spring modules: Core, Beans, Context, MVC, ORM, Data, Boot, Security, Test.

Bean wiring: specify dependencies in beans.xml

Bean wiring and autowiring?

Configure spring bean in beans.xml or context.xml file.

<bean id=”thisIsABean” class=”com.ex.HelloWorld” scope=”prototype”>

<property name=”message” value=”This is a setter injection”></property>

<constructor-arg name=”message” value=”This is constructor injection”></constructor-arg>

</bean>

@Component - register a class as a bean, allows it to be considered for auto-detection.

@Autowired - Declares a constructor, field, setter method, or configuration method to be autowired by type. Items with this annotation should be private.

* To be autowired, must configure annotations so that springs IOC container knows where to search for annotations.

Type: default(no autowiring), byName(setter injection), byType and constructor.

<context: annotation-config/>

<context: component-scan base-package=”com.ex.autowiring”>

ApplicationContext vs BeanFactory

Bean Factory: Interface, older, lazily instantitates Spring beans.

ApplicationContext: inferface that extends BeanFactory.

Bean scopes: singleton(default), prototype, request, session, global session.

Singleton: single object instance in the spring IoC container, default scope.

Prototype: many instances of the bean, new instance created whenever bean is requested from applicationContext, good for lightweight beans with conversional state.

Request – scoped to lifecycle of single HTTP request

Session – scoped to lifecycle of single HTTP session

Globalsession – scoped to lifecycle of global HTTP session

Application – scoped to lifecycle of servlet context

Web socket – scoped to lifecycle of websocket object.

Spring Bean lifecycle

1. Request bean from ApplicationContext
2. Instantiation
3. Populate properties
4. Set bean name
5. Set bean factory
6. Pre-initiation, bean post processor
7. Initializing bean
8. Custom init method
9. Post-initilization bean post processors
10. Bean is ready to use
11. Container shuts down.

Spring ORM

Spring ORM: integrate with ORM, hibernate, JPA, JDO.

Benefits: easier testing, easier transaction management, less boilerplate code, no more hibernate.cfg.xml, all configuration will be in beans.xml.

Contextual sessions:

* Hibernate code has no knowledge of Spring
* Session executes within the application context.
* Session is managed by Spring

Configure ORM in beans.xml file

* Define Spring beans: datasource, sessionfactory, transactionManager
* Inject DataSource into SessionFactory
* Inject SessionFactory into TransactionManager
* Inject SessionFactory into all DAOs.
* Use @Transactional to manage transactions.

@Transactional indicates that a persistence method takes place on DAO methods.

ORM: object relation mapping is a concept of converting the data from OOP to relational DB and vice versa.

Hibernate: implementation of ORM.

Spring Data

Spring data: is not natively compatible with hibernate, is compatible with JPA, and any vendor of an ORM tool that is JPA compliant however we need to use a vendor adapter.

What are some interfaces we use with spring data: JpaRepository

* **REQUIRED**
  + @Transactional(propagation=Propagation.REQUIRED)
  + Same physical transaction will be used if one already exists, otherwise, a new transaction will be opened.
* **REQUIRES\_NEW**
  + Indicates a new physical transaction will be created for @Transactional method --inner transition can commit or rollback independently of the outer transaction.
* **NESTED**
  + Inner and outer use same physical transaction, but are separated by savepoints (JDBC drivers only)
* **MANDATORY**
  + Existing transaction must already be opened or container will throw an error
* **NEVER**
  + Container will throw an error if a session is open (opposite of mandatory)
* **NOT\_SUPPORTED**
  + Executes outside any existing transaction, current existing transaction will be paused.
* **SUPPORTS**
  + Executes within the scope of existing transaction
  + Otherwise, executes non-transactionally

Spring MVC

* + Spring’s web module provides an implementation for the model-view-controller architectural pattern
  + Implements the front controller design pattern and uses 1 servlet that sends requests to their appropriate @Controller (or @RestController)
  + Model: Application data (Pojo’s)
  + View: Renders a view for the client using model data.
  + Controller: Processes user requests.

Process

1. User sends HTTP request to Server.
2. Single servlet is found --D.S – Dispatcher Servlet.
3. D.S. consults HandlerMapping to maps requests to correct controller.
4. D.S. sends request to controller.
5. Controller returns a response.
6. D.S. consults the view resolver.
7. View resolver used model data to render output.
8. Response is returned.

ViewResolver: helps render view, useful for server-side HTML rendering, can use any view technology: freemarker, thymeleaf, JSPs, Velocity.

Dispatcher servlet: only servlet in spring, front controller design pattern – single point of entry for all requests.

@Controller: turn java classes into beans to handle requests.

@RequestMapping: Specifies what request this method will handle, can be applied to both classes and methods.

Method: compatible HTTP methods

Params: filters requests based on presence, absence, or value of HTTP parameters

Headers: Filters requests based on presence, absence, or value of HTTP headers

Consumes:

**@PathVariable:** which URL the method is mapped to

**@ResponseBody:** return data as JSON when using @Controller, not needed when using @RestController,

**@ExeceptionHandler**

**@ResponseStatus**

**@RestController:** combines @Controller and @ResponseBody. It returns data as json.

**@RequestParam**

* + To be applied in a parameter to bind a request parameter to method parameter
  + We can specify an injected value when Spring finds no or empty

**Spring AOP**

* **Spring AOP (Aspect Oriented Programming)**

“The modularization of cross-cutting concerns”

* + Essentially injecting code that is needed in various layers of the app instead of rewriting it.
  + Affect many methods without having to change them.
* **Aspect:** classes in AOP**,** “Cross-cutting concern”**,** code to inject
  + Examples: logging, security, transactions.
* **Advice:** code to be applied.
  + 5 Types: @Before, @After, @Around, @AfterThrowing, @AfterReturning.
* **JoinPoint:** The point during code execution at which the advice is applied.
* **Pointcut:** set of join points at which a particular advice should be applied.

**Spring Boot**

* **Spring Boot Starters:** A set of convenient dependency descriptors. You will get a one-stop-shop for all the Spring and related technology that you will need without having to hunt through sample code. For example, if you want to get started using Spring and JPA for database access just include the spring-boot-starter-jpa dependency in your project.
* **@SpringBootApplication**
* **Spring** Boot **Actuator** includes a number of additional features to help you monitor and manage your application when it's pushed to production.