Introduction to Spring

/ Beans

First Spring Project

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;

@EnableAutoConfiguration
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

Bean Definition

- In Spring, 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.

IoC

- IoC is also known as dependency injection (DI).
- It is a process whereby objects define their dependencies, that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.

Bean annotations in spring

- @Component
  - Generic annotation which indicates that a java class is a bean
  - with the help of auto scan mechanism, spring picks it up and register it in the context

Bean annotations in spring (2)

- @Service
  - This annotation is basically used for classes which hold business logic in the service layer
  - It gives a logical sense that a class is a service
- @Controller
  - a stereotype used at class level in spring MVC
  - It marks a class as a spring web controller, responsible to handle HTTP request

Bean annotations in spring (3)

- @Repository
  - This annotation is used to indicate that a class is a repository:
  - Provides exception translation support for persistence operations
- @Configuration
  - Indicates that a class declares one or more @Bean methods and may be processed by the Spring container to generate bean definitions and service requests for those beans at runtimea

/ Autowired

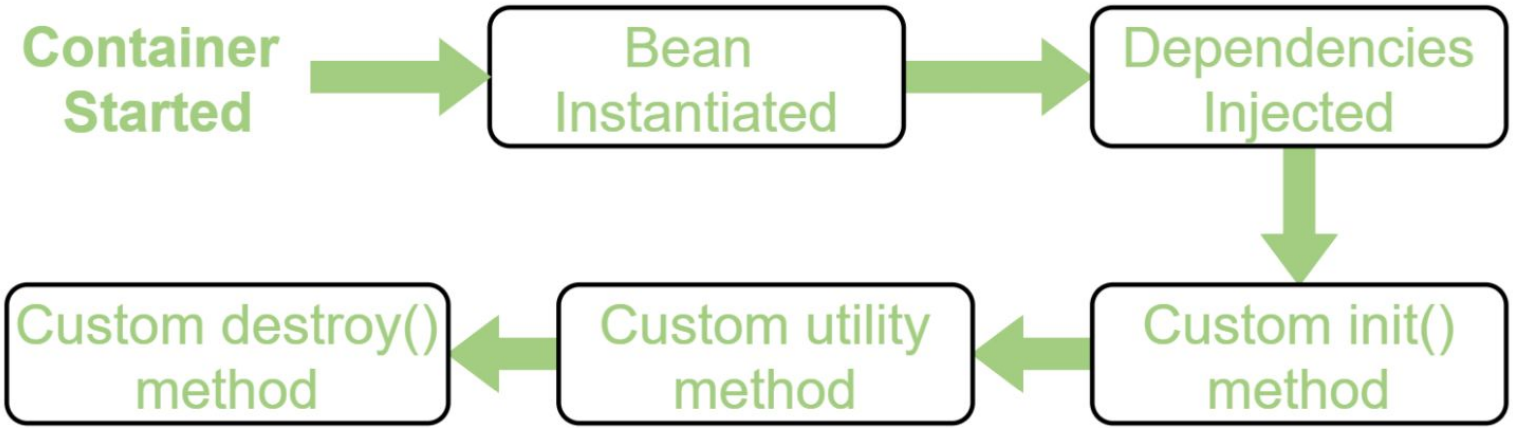
Definition

- The Spring framework enables automatic dependency injection.
- In other words, by declaring all the bean dependencies in a Spring configuration file, Spring container can autowire relationships between collaborating beans.
- This is called Spring bean autowiring.

Types of Autowire

- @Autowired on Properties
- @Autowired on Setters
- @Autowired on Constructor

/ Beans Life Cycle


Container Started
Bean Instantiated
Dependencies Injected
Custom init() method
Custom utility method
Custom destroy() method

Moments in Life Cycle

1. Bean Initialization -> Constructor

1. Post Initialization
    - @PostConstruct
    - Configured init() method

1. Pre Destroy
    - @PreDestroy
    - Configured destroy() method

/ Scope of beans

Scope of the beans

1. Singleton
2. Prototype
3. Request
4. Session
5. Application
6. Websocket

Singleton beans

- This scope is the default value if no other scope is specified, but can be also defined like this:
  - `@Scope("singleton")`
  - `@Scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)`
- All requests for that bean name will return the same object
- Any modifications to the object will be reflected in all references to the bean

```
@Bean
@Scope("singleton")
public Person personSingleton() {
    return new Person();
}
```

Prototype beans

- A bean with the prototype scope will return a different instance every time it is requested from the container
- Can be define:
  - `@Scope("prototype")`
  - `@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)`

Session beans

- Is a scope that's used in the web-aware Application Context.
- It's created per every HTTP session and is present during the whole HTTP session lifecycle.
- Can be define:
  - `@SessionScope`
  - `@Scope(value = WebApplicationContext.SCOPE_SESSION, proxyMode = ScopedProxyMode.TARGET_CLASS)`

/ @Profile

How @Profile works

- The @Profile annotation is used for mapping the bean to that particular profile
- The annotation simply associate the names of one (or multiple) profiles to a bean
- Basic scenario:
  - We have a bean that should only be active during development but not deployed in production
  - We want to change the db type on the fly

Activate Profile

- From code

```
@Configuration
public class MyWebApplicationInitializer
  implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {

        servletContext.setInitParameter(
          "spring.profiles.active", "dev");
    }
}
```

- From arguments

```
-Dspring.profiles.active=dev
```

/ Multiple Beans

Multiple Beans present

- By default, Spring resolves @Autowired entries by type.
- If no bean of that type is available in the container, the framework will throw a fatal exception.
- If more than one bean of the same type is available in the container, the framework will throw a fatal exception.
- Also can be autowired all the list that are compatible with that class

@Qualifier

- By using the @Qualifier annotation, we can eliminate the issue of which bean needs to be injected.
- By including the @Qualifier annotation, together with the name of the specific implementation we want to use, we can avoid ambiguity when Spring finds multiple beans of the same type.

@Primary

- There's another annotation called @Primary that we can use to decide which bean to inject when ambiguity is present regarding dependency injection.
- This annotation defines a preference when multiple beans of the same type are present. The bean associated with the @Primary annotation will be used unless otherwise indicated.

/ Thymeleaf

Templates

- resources/templates – HTML files
- Technologies: JSP (Java Server Pages), Thymeleaf , Groovy, Jade etc.
- Thymeleaf
  - Similar in syntax with HTML
  - Support for defining forms behavior
  - Binding form inputs to data models
  - Validation for form inputs
  - Displaying values from message sources
  - Rendering template fragments

ModelAndView

- Interface for passing values to a template (view)

```
@GetMapping("/view")
public ModelAndView displayView() {
    ModelAndView modelAndView = new ModelAndView("viewPage");
    modelAndView.addObject("message", “abc");
    return modelAndView;
}
```

Thymeleaf expressions

- Thymeleaf expressions (1)${...} : Variable expressions. These are OGNL expressions (or model attributes in Spring)
- *{...} : Selection expressions, it will be executed on a previously selected object only
- #{...} : Message (i18n) expressions. Used to retrieve locale - specific messages from external sources
- @{...} : Link (URL) expressions. Used to build URLs
- ~{...} : Fragment expressions. Represent fragments of markup and move them around templates

Thymeleaf example

```
<td th:text="${teacher.additionalSkills} ?: 'UNKNOWN'" /> (Elvis operator)

<td th:text="${teacher.active} ? 'ACTIVE' : 'RETIRED'" />

<td>
    <span th:if="${teacher.gender == 'F'}">Female</span>
    <span th:unless="${teacher.gender == 'F'}">Male</span>
</td>
```

/ Rest API

What is an API

- API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.
- Each time you use an app like Facebook, send an instant message, or check the weather on your phone, you’re using an API.

Rest API

- A REST API (also known as RESTful API) is an application programming interface (API or web API) that conforms to the constraints of REST architectural style and allows for interaction with RESTful web services.
- REST is a set of architectural constraints, not a protocol or a standard. API developers can implement REST in a variety of ways. When a client request is made via a RESTful API, it transfers a representation of the state of the resource to the requester or endpoint.
- This information, or representation, is delivered in one of several formats via HTTP: JSON (Javascript Object Notation), HTML, XLT, Python, PHP, or plain text. JSON is the most generally popular file format to use because, despite its name, it’s language-agnostic, as well as readable by both humans and machines.

/ Q&A


OBILE/ACADEMY