What's new in Spring 5 ?

At a high level, features of Spring Framework 5.0 can be categorized into :

- JDK baseline update (Works with Java 8 and more).
- Core framework revision (Based on Java 8).
- Core container updates.
- Functional programming with Kotlin.
- Reactive Programming Model.
- Testing improvements.
- Library support.
- Discontinued support.

More details :

https://www.udemy.com/course/spring-framework-5-beginner-to-guru/learn/lecture/7577382#overview

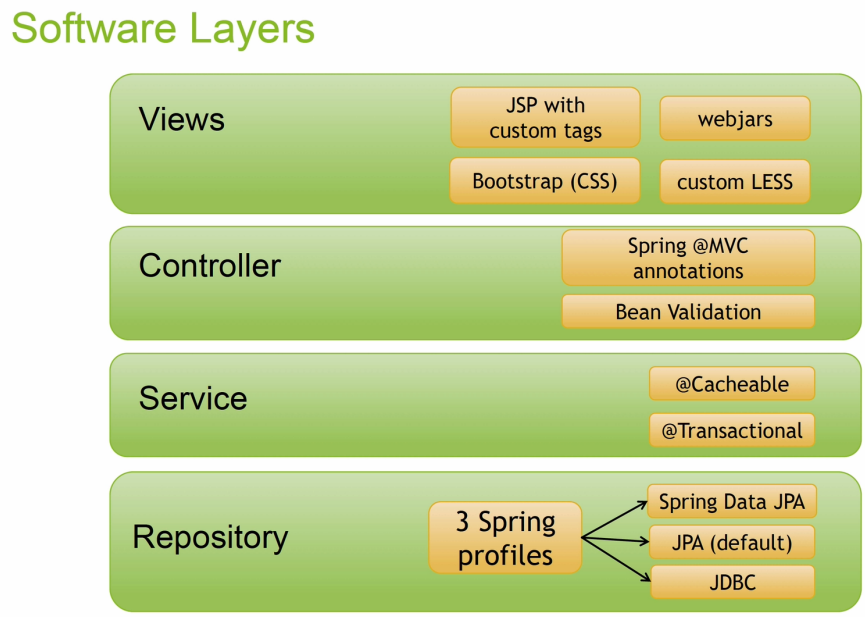


Figure 1: Software Layers

Spring Boot :

We can see it as a wrapper over Spring Framework that helps with configuration of Spring applications.

Spring Boot Starters : A list of projects with the necessary dependencies for each type of starter.

https://github.com/spring-projects/spring-boot/tree/main/spring-boot-project/spring-boot-starters

https://www.baeldung.com/spring-boot-starters

C'est du Tomcat sous le capot :

```
2023-06-21 15:04:51.988  INFO 26892 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer :
    Tomcat started on port(s): 8080 (http) with context path ''
```

CommandLineRunner : Needs to implement a run method which is executed when Spring Application is launched.

Difference between Spring and Spring Boot :

https://stacklima.com/difference-entre-spring-et-spring-boot/

JPA :

Spring Data Repositories : By extending CrudRepository it gives access to several CRUD methods (save, findById, delete, count...).

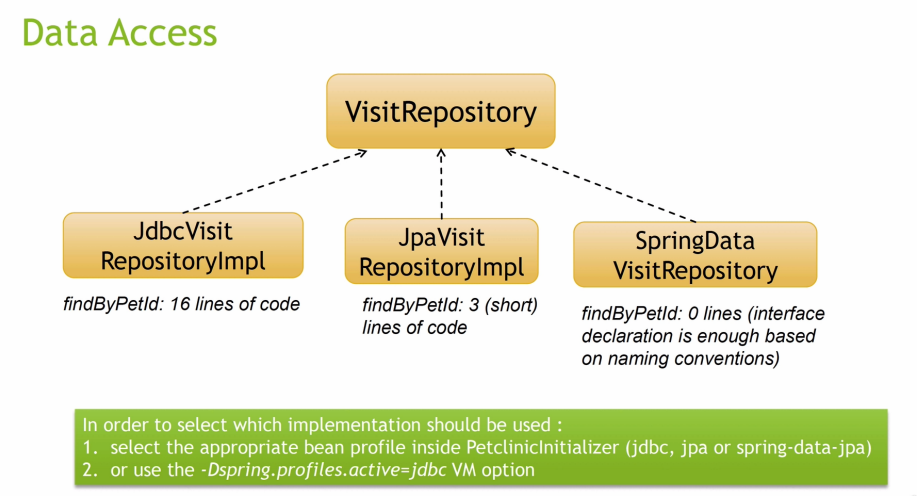


Figure 2: Data Access

@Entity : Declares a class as a JPA entity.

@Id : The Id property is mapped to the primary key in the underlying database. Thus it is used to persist and retrieve object from the database.

Hibernate will generate SQL based on the JPA definition.

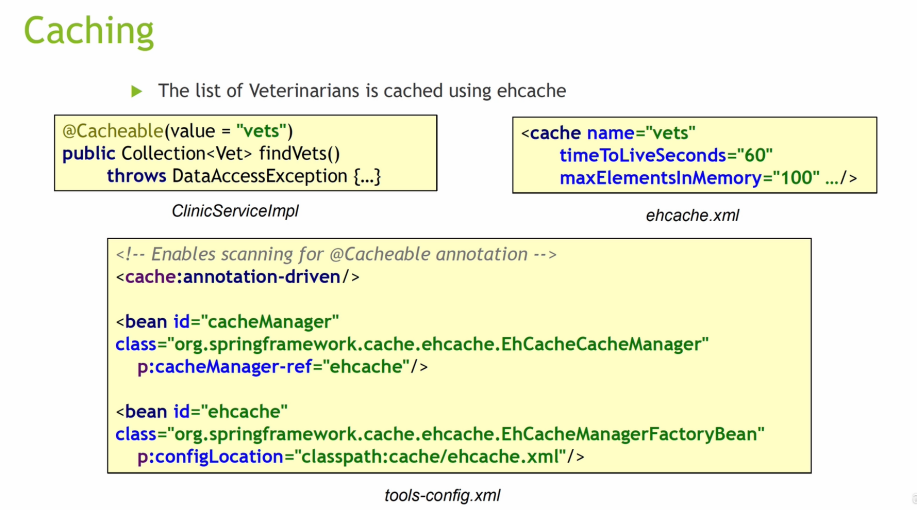


Figure 3: Caching

Spring MVC :

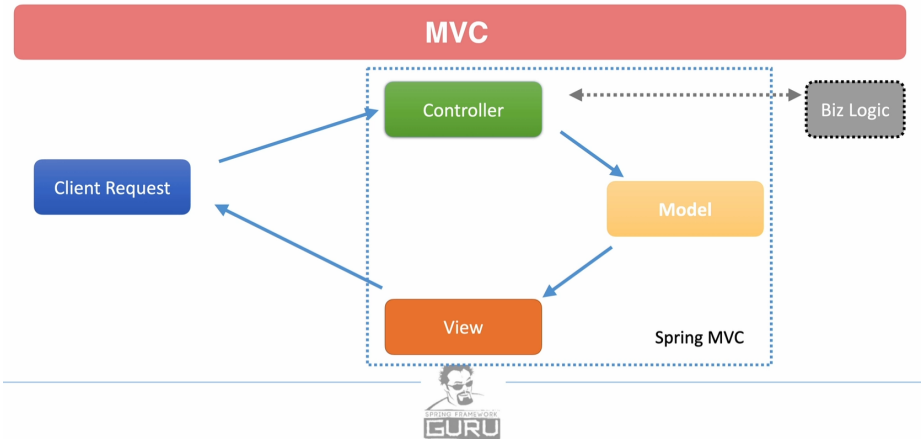


Figure 4: MVC

@Controller : Will register the class as a Spring Bean and as a Controller in Spring MVC.

To map controller methods to HTTP request paths use @RequestMapping.

@ResponseBody : To return simple stream without any template or view resolving.

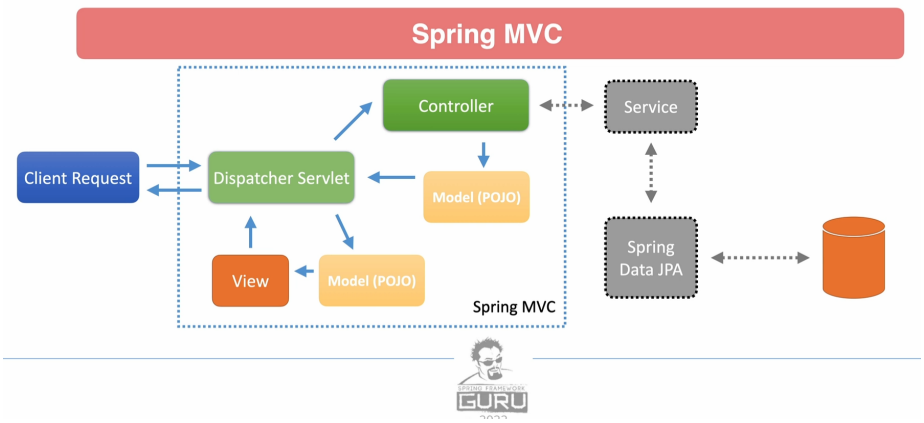


Figure 5: Spring MVC

Thymeleaf : Java template engine, an alternative to JSP. It is a natural template engine. Natural meaning that templates can be viewed in a browser.

Spring Dependency Injection :

IoC : It is a technique to allow dependencies to be injected by runtime. It is up to Sring Framework to manage the construction of the beans. It allows the framework to compose the application by controlling which implementation is injected (For example : H2 database datasource or mySQL datasource).

Purpose : The framework plays the role os the main program in coordinating and sequencing application activity. This inversion of Control gives the power to serve as extensible skeleton. The methods supplied by the user tailor the generic algorithms defined in the framework for a particular application. Developer will focus development efforts on the application code and framework will assemble application.

Spring Context : Is the stuff around the application and when Spring starts up it will look into the project for annotated dependencies or configuration files. As it finds Spring Beans, components they are going to be constructed and become available in the context. Then we can ask the context, Spring Application Context, for a reference to that object. You don't have to manage all the objects by using dependency management.

Types of Dependency Injection :

- By class properties : Can be public or private. If private Spring can use reflection to set. Works but it is slow and makes testing difficult.
- By setters : Most used. Spring had in the past some issues with DI by constructors.
- By Constructor : Most preferred.

Another tip : Use final properties for injected components.

Dependency Injection can be done with Interfaces and it is highly preferred :

- It allows runtime to decide implementation to inject.
- It follows Interface Segregation Principle of SOLID.
- It makes code more testable - mocking becomes trivial.

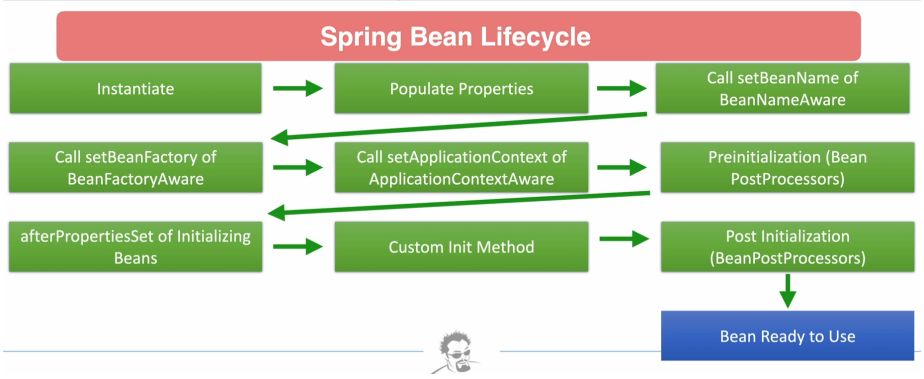
@Autowired annotation not needed for Constructor Injection.

The **@Qualifier** annotation will take precedence over Primary.

Profiles are one of the most important features of the Spring Framework.

Profiles allow to have beans in a configuration that will take different characteristics depending on the environment. Like for example a datasource that can be H2 on dev environment and mySQL on production.

Spring Bean Life Cycle :



Call back Interfaces that can be implemented for call back events :

- InitilizeBean.afterPropertiesSet() : Called after properties are set.
- DisposableBean.destroy() : Called during destruction in shutdown.

Lifecycle Annotations :

- @PostConstruct annotated methods will called after the has been constructed.
- @PreDestroy is called just before the bean is destroyed by the container.

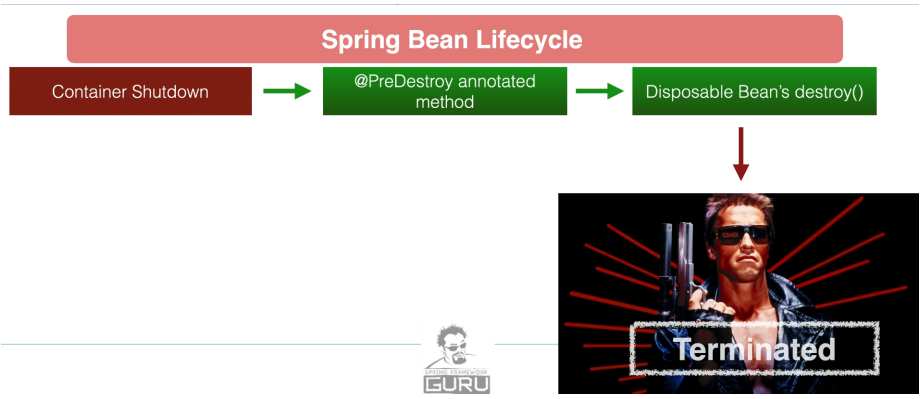


Figure 6: Spring Bean Lifecycle

Bean Post Processors : Give the mean to tap into the Spring context life cycle and interact with beans as they are processed. It is called for all beans in context.

Aware Interfaces : They are used to access the Spring Framework infrastructure. They are largely used within the framework but rarely by developers.

SOLID ?

Open Closed Principle (for O) :

https://www.udemy.com/course/spring-framework-5-beginner-to-guru/learn/lecture/17855112#questions

Interface Segregation Principle (for I) :

https://www.udemy.com/course/spring-framework-5-beginner-to-guru/learn/lecture/17855124#questions

Dependency Inversion Principle :

https://www.udemy.com/course/spring-framework-5-beginner-to-guru/learn/lecture/17855182#questions

---

Spring Configuration Options :

XML Based Configuration :

- Introduced in Spring Framework 2.0.
- Common in legacy Spring Applications.

- Newer Spring Applications use Java configuration options.
- Still supported in Spring Framework 5.x : @importResource("classpah:sfgdi-config.xml") on Configuration class.

```
@importResource("classpah:sfgdi-config.xml")
@Configuration
public class GreetingServiceConfig {
        ...
```

Annotation Based Configuration - Annotations introduced in Java 5 :

- Introduced in Spring Framework 3.
- Spring beans found via 'Component Scans' : A scan of packages for annotated components.
- Spring Beans are found via class level annotations : @Controller, @Service, @Component, @Repository.

Java Based Configuration :

- Introduced in Spring Framework 3.0.
- Uses Java classes to define Spring Beans.
- Configuration classes are defined with @Configuration annotation.
- Methods returning Spring Beans declared with @Bean annotation.

Which to Use ?

- You can use one or a combination of methods.
- All will work seamless together to define beans in the Spring Context.
- While XML is still widely used with legacy applications, industry trend is to Java based.
- Ideally legacy applications should be migrated to Java based configuration.
- Newer applications should use Java based configuration.

Spring Framework Stereotypes :

- Stereotype - A fixed general image or set of characteristics which represent a particular type of person or thing.
- Spring Stereotypes are class level annotations used to define Spring Beans. When classes annotated with Spring Stereotypes are detected via the component scan, an instance of the class will be added to the Spring context.

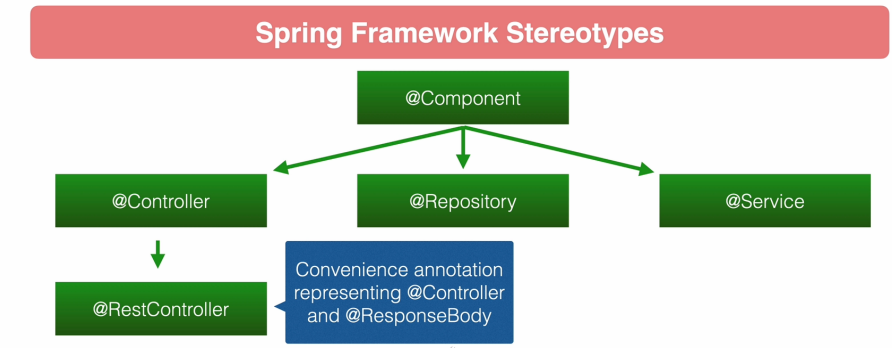


Figure 7: Spring Framework Stereotypes

- Available Spring Stereotypes : @Component, @Controller, @RestController, @Repository, @Service.

- **@Component :** Indicates that an annotated class is a "component" and it will be created as a bean.
- **@Controller :** Indicates that an annotated class has the role of a Spring MVC "Controller".
- **@RestController :** Convenience Annotation which extends @Controller, and adds @ResponseBody.
- **@Repository :** Indicates class is a "Repository", originally defined by Domain-Driven Design (Evans, 2003) as "a mechanism for encapsulating storage, retrieval, and search behavior which emulates a collection of objects".
- **@Service :** Indicates that an annotated class is a "Service", originally defined by Domain-Driven Design (Evans, 2003) as "an operation offered as an interface that stands alone in the model, with no encapsulated state.".

Spring Component Scan :

- Spring Beans defined with Spring Stereotypes are detected with Spring component scan.
- On startup Spring is told to scan packages for classes with Spring Stereotype annotations.
- This configuration is Spring Framework specific, NOT Spring Boot.
- Spring Boot's auto configuration will tell Spring to perform a component scan of the package of the main class. This includes all sub packages of the main class package.

- When using Spring Boot, if class is outside of the main class package tree, you must declare the package scan.

Componant Scan : Is performed within the package where the application SpringBootApplication is located. To look outside of the default package :

```
@ComponentScan(basePackages = { "guru.springframework.sfgdi",
                                "com.springframework.pets" })
```

Spring uses reflection to instanciate components, it can be slow to perform component scan for application with thousands of components. Then refactor the application to use Java configuration rather than component scan and reflection on startup.

@Configuration tells Spring Framework that this is a Spring configuration class (detected by component scan :)).

@Bean instead of Component can be useful when working with a third party component. When using some type of library, like Jackson for JSON processing. The name of the bean in the Spring Context will the name of the method that creates it within the configuration class.

So if you own the code use stereotypes but if you don't own the code and you wnat to bring that objects into Spring Context, use configuration classes.

Spring Bean Scopes :

- Singleton - Only one instance of the bean is created in the IoC container (default). They are created at the Spring Context startup.
- Prototype - A new instance is created each time the bean is requested. They are created on demand.
- Request - A single instance per http request. Only valid in the context of a web-aware Spring ApplicationContext.
- Session - A single instance per http session. Only valid in the context of a web-aware Spring ApplicationContext.
- Global-Session - A single instance per global session. Typically only used in a Portlet context. Only valid in the context of a web-aware Spring ApplicationContext.
- Application - bean is scoped to the lifecycle of a ServletContext. Only valid in the context of a web aware.
- Websocket - Scopes a single bean definition to the lifecycle of a WebSocket. Only valid in the context of a web-aware Spring ApplicationContext.
- Custom Scope - Spring Scopes are extensible, and you can define your own scope by implementing Spring's "Scope" interface.

Tips :

- No declaration needed for singleton scope.
- In Java configuration use @Scope annotation.
- In XML configuration scope is an XML attribute of the 'bean' tag.
- 99% of the time singleton scope is fine!

External Properties with Spring Framework :

They can be set through Command Line Arguments, JSON obejct via command line or environment variable, JNDI, OS environment variables, properties files (.properties, YAML)...

- Properties can be overridden depending on how they are defined.
- Gives you a lot of flexibility for deployments.
- Lowest are properties defined in JAR or war properties or YAML files.
- Next are external property files to JAR via file system.
- Higher are profile specific properties files (in jar, then external).
- OS Environment variables.

Property Hierarchy :

1. Java system properties (lowest).
2. JNDI.
3. SPRING_APPLICATION_JSON.
4. Command Line Arguments.
5. Test Properties (for testing).

- Favor using application.properties or application.yml in packaged JAR (or WAR).

```
@PropertySource("classpath:datasource.properties")
@Configuration
public class GreetingServiceConfig {

    @Bean
    FakeDataSource fakeDataSource(@Value("${guru.username}") String username,
                                  @Value("${guru.password}") String password,
                                  @Value("${guru.jdbcurl}") String jdbcurl) {
        ...
    }
```

It is possible to use profiles to pick convenient properties file :

```
$ application.properties
spring.profiles.active=cat,EN,dev

$ application-dev.properties
$ application-qa.properties
```

Properties Binding :

```
@Configuration
@ConfigurationProperties("guru")
public class SfgConfiguration {

    // Will look for the property in application.yml guru section, thanks to ConfigurationPr
    private String username;
```

Example of Application :

Comparing with the original Spring Petclinic

	Spring Framework Petclinic	« Canonical » Spring Petclinic
Spring stack	Plain Old Spring Framework	Spring Boot
Architecture	3 layers	Aggregate-oriented domain
Persistence	JDBC, JPA, Spring Data JPA	Spring Data JPA
View	JSP	Thymeleaf
Databases support	HSQLDB, MySQL, PostgreSQL	HSQLDB, MySQL
Containers support	Tomcat 7 and 8, Jetty 9	Embbeded Tomcat and Jetty
Java support	Java 7 and 8	Java 8

- « Canonical » implementation : https://github.com/spring-projects/spring-petclinic
- Spring Framework version : https://github.com/spring-petclinic/spring-framework-petclinic

Figure 8: Pet Clinic Application