Java Web module – Theory

All in one

Q: What is Spring IOC Container?

A: At the core of the Spring Framework, lies the Spring container. The container creates the object, wires them together, configures them, and manages their complete life cycle. The Spring container makes use of Dependency Injection to control the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations, or Java code.

Q: What do you mean by Dependency Injection?

A: In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don't connect your components and services in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

Q: Spring Beans

A: They are the objects that form the backbone of the user's application. Beans are managed by the Spring IoC container. They are instantiated, configured, wired, and managed by a Spring IoC container. Beans are created with the configuration metadata that the users supply to the container.

Q: Bean scopes

A: Singleton: This provides scope for the bean definition to a single instance per Spring IoC container.

Prototype: This provides scope for a single bean definition to have any number of object instances.

Request: This provides scope for a bean definition to an HTTP request.

Session: This provides scope for a bean definition to an HTTP session.

Global-session: This provides scope for a bean definition of a Global HTTP session.

Q: What's the difference between @Component, @Controller, @Repository & @Service annotations in Spring?

A: @Component: This marks a java class as a bean. It is a generic stereotype for any Spring-managed component. The component-scanning mechanism Spring now can pick it up and pull it into the application context.

@Controller: This marks a class as a Spring Web MVC controller. Beans marked with it are automatically imported into the Dependency Injection container.

@Service: This annotation is a specialization of the Component annotation. It doesn't provide any additional behavior over the @Component annotation. You can use @Service over @Component in service-layer classes as it specifies intent in a better way.

@Repository: This annotation is a specialization of the @Component annotation with similar use and functionality. It provides additional benefits specifically for DAOs. It imports the DAOs into the DI container and makes the unchecked exceptions eligible for translation into Spring DataAccessException.

Q: What do you understand by @Qualifier annotation?

A: When you create more than one bean of the same type and want to wire only one of them with a property you can use the @Qualifier annotation along with @Autowired to remove the ambiguity by specifying which exact bean should be wired.

Q: What do you understand by @RequestMapping annotation?

A: @RequestMapping annotation is used for mapping a particular HTTP request method to a specific class/ method in the controller that will be handling the respective request. This annotation can be applied at both levels:

Class level: Maps the URL of the request

Method level: Maps the URL as well as the HTTP request method.

Q: Describe AOP Aspect-Oriented Programming.

A: Aspect-oriented programming or AOP is a programming technique that allows programmers to modularize crosscutting concerns or behavior that cuts across the typical divisions of responsibility. Examples of cross-cutting concerns can be logging and transaction management. The core of AOP is an aspect. It encapsulates behaviors that can affect multiple classes into reusable modules.

Q: Describe some standard Spring Events?

A: Spring provides the following standard events:

Context Refreshed Event: This event is published when the Application Context is either initialized or refreshed. This can also be raised using the refresh () method on the Configurable Application Context interface.

Context Started Event: This event is published when the Application Context is started using the start () method on the Configurable Application Context interface. The user can poll their database or they can re/start any stopped application after receiving this event.

Context Stopped Event: This event is published when the Application Context is stopped using the stop () method on the Configurable Application Context interface. The users can do the necessary housekeeping work after receiving this event.

Context Closed Event: This event is published when the Application Context is closed using the close () method on the Configurable Application Context interface. A closed context reaches its end of life; it cannot be refreshed or restarted.

Request Handled Event: This is a web-specific event telling all beans that an HTTP request has been serviced.

Q: Name Some of the Design Patterns Used in the Spring Framework?

A: Singleton Pattern: Singleton-scoped beans

Factory Pattern: Bean Factory classes

Prototype Pattern: Prototype-scoped beans

Adapter Pattern: Spring Web and Spring MVC

Proxy Pattern: Spring Aspect Oriented Programming support

Template Method Pattern: JdbcTemplate, HibernateTemplate, etc.

Front Controller: Spring MVC DispatcherServlet

Data Access Object: Spring DAO support

Model View Controller: Spring MVC

Q: Which is good to use – Constructor or Setter-based dependency?

A: Constructor Injection is better than other patterns from all other patterns. Both types of dependency injection can be used accordingly based on the situation. It is a thumb rule, that for mandatory dependency, constructor-based dependency injection is used while for optional dependency, setter-based dependency injection is used.

It is advisable to utilize constructor injection for every mandatory collaborator and setter injection for every single other property. Once more, constructor injection guarantees every required property has been fulfilled, and it is essentially impractical to instantiate an object in an invalid state (not having passed its collaborators). When utilizing constructor injection you don't need to utilize a dedicated mechanism to guarantee required properties are set (other than a typical Java mechanism).

In case we are using field injection which is non-final, they are prone to circular dependencies. However, it exposes coupling if there are more than 3 objects as part of the constructor injection. It is also easy to test if we use constructor injection.

Q: IOC vs Dependency Injection

A: IOC: Inversion of control is used when you are not worried about the actual creation of objects. You will assume that the object will be created and it will be supplied to the executor while execution. Inversion of Control is a principle in software engineering by which the control of objects or portions of a program is transferred to a container or framework. It's most often used in the context of object-oriented programming. IOC can be implemented using service locators, events, delegates, or dependency injection.

Dependency Injection is a form of IOC. Dependency injection can be achieved via constructor injection, property injection, or even a method injection. DI provides a way to inject objects into other objects.

Q: Which design patterns are used in the Spring framework?

A: Design patterns help to follow good practices of programming. Spring framework, as one of the most popular web frameworks, also uses some of them.

The proxy pattern is used heavily in AOP and remoting.

Spring class "org.springframework.aop.framework.ProxyFactoryBean" uses a proxy design pattern. It builds the AOP proxy based on Spring beans.

The proxy provides a placeholder for another bean to control access to it.

In the Singleton design pattern, only a single instance of the class is returned on any number of requests to create an object of the class. In the spring framework, the Singleton is the default scope and the spring container creates and returns an exactly single instance of the class per spring container. Spring container cache the singleton bean and return the same bean upon any request for the bean. It is recommended to use the singleton scope for stateless beans.

Factory design pattern

It provides a public static factory method to initialize the object.

The Spring framework uses the factory design pattern to create the bean using BeanFactory Container and ApplicationContext Container.

Template Design Pattern

The pattern is used heavily to reduce boilerplate code. For example JdbcTemplate, JmsTemplate, and JpaTemplate.

Model View Controller Pattern

Spring MVC is using the pattern in the web application. The controller is POJO instead of the servlet, which makes controller testing easier. The controller returns the logical view name and the responsibility to resolve the view is on ViewResolver, which makes it easier to reuse the controller for different view technologies.

Front Controller Pattern

Spring dispatcher servlet dispatches all incoming requests to the mapped controller. It helps to implement all cross-cutting concerns or tracking requests and responses.

Dependency injection or inversion of control (IOC)

The spring framework is responsible for the creation, writing, and configuring of objects and manages the entire lifecycle of these objects until they are destroyed. The container has the Dependency Injection (DI) responsible to manage the components present in an application.

HTTP

Q: Hyper Text Transfer Protocol - HTTP

A: HTTP defines methods to indicate the desired action to be performed on the identified resource.

GET

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data.

HEAD

The HEAD method asks for a response identical to that of a GET request, but without the response body.

POST

The POST method is used to submit an entity to the specified resource, often causing a change in state or side effects on the server.

PUT

The PUT method replaces all current representations of the target resource with the request payload.

DELETE

The DELETE method deletes the specified resource.

CONNECT

The CONNECT method establishes a tunnel to the server identified by the target resource.

OPTIONS

The OPTIONS method is used to describe the communication options for the target resource.

TRACE

The TRACE method performs a message loop-back test along the path to the target resource.

PATCH

The PATCH method is used to apply partial modifications to a resource.

Q: HTTP Response Codes

A: HTTP response code classes

1xx: informational (e.g., ""100 Continue"")

2xx: successful (e.g., ""200 OK"", ""201 Created"")

3xx: redirection (e.g., ""304 Not Modified"", ""301 Moved Permanently"", ""302 Found"")

4xx: client error (e.g., ""400 Bad Request"", ""404 Not Found"", ""401 Unauthorized"", ""409 Conflict"")

5xx: server error (e.g., ""500 Internal Server Error"", ""503 Service Unavailable"")

Cookies

Q: What Are Cookies?

A: A small file of plain text with no executable code. Sent by the server to the client's browser. Stored by the browser on the client's device (computer, tablet, etc.). Hold a small piece of data for a particular client and a website.

Q: What Are Cookies Used for?

A: Session management - Logins, shopping carts, game scores, or anything else the server should remember.

Personalization - User preferences, themes, and other custom settings.

Tracking - Recording and analyzing user behavior.

Error Handling

Q: Controller-Based Error Handling

A: You can define Controller-specific Exception Handlers.

Annotated with @ExceptionHandler annotation.

They work only for the Controller they are defined in.

Can be annotated with @ResponseStatus to convert HTTP status.

Can accept the caught exception as a parameter.

Can return ModelAndView or String (view name).

Can catch multiple exception types.

In Controller-Based Error handling we can inject - HttpServletRequest, HttpServletResponse, HttpSession, Principal.

Q: Global Exception Handling

A: There is a way to achieve global exception handling in Spring

This is done through the @ControllerAdvise annotation

Any class annotated with @ControllerAdvise turns into an interceptor-like controller.

In @ControllerAdvice classes you still use @ExceptionHandler.

However, this time it refers to the whole application.

The error handling is not limited only to a specific controller.

Global Exception Handling (REST).

HTTP Error response codes are a good choice.

However, sometimes you might need more than just a status.

Customized Error Object, which can be presented to the Client.

Limited Information was returned to the Client.

Thymeleaf

Q: What is Thymeleaf?

A: Thymeleaf is a modern server-side Java template engine used in Spring.

It allows us to:

Use variables in our views.

Execute operations on our variables.

Iterate over collections.

Make our views dynamical.

Q: Thymeleaf Standard Expressions

A: Variable Expressions - ${...}

Link(URL) Expressions - @{...}

Selection Expressions - *{...}

Fragment Expressions - ~{...}

Accession Bean - ${@...}

Events in Spring

Q: Scheduling Tasks

A: Scheduling is a process of executing tasks for a specific period.

Spring Boot provides good support to write a scheduler on the Spring applications.

We can specify the period in different ways:

Using Cron.

Using Fixed Rate is used to execute the tasks at a specific time.

It does not wait for the completion of the previous task.

The values should be in milliseconds.

Using Fixed Delay is the time between tasks.

The initialDelay is the time after which the task will be executed the first time after the initial delay value.

It waits for the completion of the previous task.

The @EnableScheduling annotation is used to enable the scheduler for your application.

Q: Caching Data

A: Caching

When using Spring Boot, the @EnableCaching annotation would register the ConcurrentMapCacheManager.

Use @Cacheable to demarcate cacheable methods.

The result is stored in the cache and on subsequent invocations (with the same arguments), the value in the cache is returned without having to execute the method.

Custom Cache Resolution -> cacheManager = "myCacheManager"

Conditional Caching -> condition = "#avg > 4"

@CachePut

When the cache needs to be updated without interfering with the method execution.

The method is always executed, and its result is placed into the cache.

@CacheEvict

This process is useful for removing stale or unused data from the cache.

Using the allEntries attribute to evict all entries from the cache.

Security

Q: Interceptor

A: Spring Interceptors are defined in the Spring context.

The interceptor includes three main methods:

preHandle: executed before the execution of the target resource

afterCompletion: executed after the execution of the target resource (after rendering the view)

postHandle: Intercept the execution of a handler

To use interceptors we need to register them:

```
public class LogingInteceptor implements HandlerInterceptor {
   @Override
   public boolean preHandle(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain, Object handler) throws IOException, ServletException {
 //Log some information ... retrun true; } }
```

Register in Configuration:

```
@Configuration
@EnableWebMvc
public class WebConfiguration implements WebMvcConfigurer {
   private final MyInterceptor myInterceptor;
   public WebConfiguration(MyInterceptor myInterceptor){this.myInterceptor = myInterceptor;}
   @Override
   public void addInterceptors(InterceptorRegistry registry) {
      registry.addInterceptor(myInterceptor);}}
```

Q: What is the principal?

A: The principal is just an Object. Most of the time this can be cast into a UserDetails object. UserDetails is a central interface in Spring Security. It represents a principal but in an extensible and application-specific way. Think of UserDetails as the adapter between your user database and what Spring Security needs inside the SecurityContextHolder. Being a representation of something from your user database, quite often you will cast the UserDetails to the original object that your application provided, so you can call business-specific methods (like getEmail(), getEmployeeNumber(), and so on).

Testing

Q: Unit Testing

A: A level of software testing where individual components are tested.

The purpose is to validate that each unit performs as designed.

The lowest level of software testing.

Often isolated to ensure individual testing.

Q: Mocking

A: Software practice, primarily used in Unit Testing.

An object under test may have dependencies on other objects.

To isolate the behavior, the other objects are replaced.

The replacements are mocked objects.

The mocked objects simulate the behavior of the real objects.