ItsRunTym

JAVA Spring

60 interview questions/answers

Topics Included:

1. **Spring Core Concepts**
   - Inversion of Control (IoC) and Dependency Injection (DI)
   - Bean lifecycle and scopes
   - Stereotype annotations (@Component, @Service, etc.)
   - Spring configuration (@Configuration, @Bean)
2. **Spring AOP (Aspect-Oriented Programming)**
   - Concepts of AOP
   - Defining aspects with @Aspect, pointcuts, and advices (@Before, @After, @Around)
   - Use cases for AOP in Spring
3. **Spring MVC (Model-View-Controller)**
   - Components of Spring MVC (@Controller, @RequestMapping, etc.)
   - REST API creation and handling
   - @RestController, @GetMapping, @PostMapping
   - Exception handling in Spring MVC (@ExceptionHandler, @ControllerAdvice)
4. **Spring Boot**
   - Auto-configuration and embedded server
   - @SpringBootApplication and application.properties
   - Profiles and environment-based configurations
   - DevTools, Actuator, and CLI usage
5. **Spring Data JPA**
   - Spring Data repositories (CrudRepository, JpaRepository)
   - Entity annotations (@Entity, @Id, @GeneratedValue)
   - Query methods and @Query annotation
6. **Spring Transactions**

   - @Transactional and transaction management
   - Isolation levels and propagation settings

7. **Spring Security**
   - Basic authentication and authorization
   - Method-level security (@PreAuthorize, @PostAuthorize)
   - CSRF protection
   - OAuth2 and JWT (JSON Web Token)
8. **Spring Cloud**
   - Client-side load balancing with Ribbon
   - Spring Cloud Config for centralized configuration management
   - Hystrix for circuit breaking
   - Spring Cloud Gateway for routing requests
   - Spring Cloud Stream for messaging between microservices

Spring Core Concepts

1. **What is Inversion of Control (IoC) in Spring? How does it differ from Dependency Injection (DI)?**

Answer: Inversion of Control (IoC) is a design principle in which the control over object creation and dependency resolution is transferred from the application to the Spring framework. Dependency Injection (DI) is a specific implementation of IoC in which dependencies are injected into an object rather than being created by the object itself.

2. **Explain the types of Dependency Injection in Spring.**

Answer: Spring supports two types of Dependency Injection:

   - **Constructor-based DI**: Dependencies are provided through a class constructor.
   - **Setter-based DI**: Dependencies are injected through setter methods after object creation.

Example:

java

```
@Component
public class Service {
  private Repository repository;
```

```
    @Autowired
    public Service(Repository repository) { // Constructor-based DI
        this.repository = repository;
    }

    @Autowired
    public void setRepository(Repository repository) { // Setter-based DI
        this.repository = repository;
    }
}
```

3. **Describe the lifecycle of a Spring bean.**

Answer: The bean lifecycle in Spring involves several stages:

- **Instantiation**: Bean is created.
- **Property Population**: Dependencies are injected.
- **Initialization**: Any custom initialization (using @PostConstruct or InitializingBean).
- **Usage**: Bean is ready to be used.
- **Destruction**: Bean is destroyed when the container shuts down, with optional cleanup (using @PreDestroy or DisposableBean).

4. **What are Spring bean scopes? Name the commonly used scopes.**

Answer: Spring bean scopes define the lifecycle and visibility of beans in the Spring container. Common scopes include:

- **Singleton**: Single instance per container (default).
- **Prototype**: New instance for each request.
- **Request**: New instance per HTTP request (web applications).
- **Session**: New instance per HTTP session (web applications).

5. **What are stereotype annotations in Spring? List some examples.**

Answer: Stereotype annotations are used to mark classes as Spring-managed components, making it easier to configure and autowire beans. Common stereotype annotations include:

- @Component: General purpose.
- @Service: Business logic layer.
- @Repository: Data access layer.

- @Controller: Web controller in MVC.

6. **What is the purpose of the @Configuration and @Bean annotations in Spring?**

Answer: @Configuration is used to define a configuration class, which contains bean definitions. @Bean is used within a @Configuration class to define and configure beans managed by the Spring container.

Example:

java

```
@Configuration

public class AppConfig {

  @Bean

  public MyService myService() {

    return new MyServiceImpl();

  }

}
```

7. **What is autowiring in Spring, and how does it work?**

Answer: Autowiring in Spring automatically injects dependencies by matching types, qualifiers, or names. It can be configured using annotations like @Autowired for field, constructor, or setter injection.

Spring AOP (Aspect-Oriented Programming)

8. **What is Aspect-Oriented Programming (AOP), and why is it used in Spring?**

Answer: AOP is a programming paradigm that allows separating cross-cutting concerns (e.g., logging, transaction management) from core business logic. In Spring, AOP enables developers to modularize these concerns into aspects, simplifying code and reducing repetition.

9. **What are the main components of Spring AOP?**

Answer:

- **Aspect**: A module containing cross-cutting logic.
- **Join Point**: A point in the program where an aspect can be applied (e.g., method execution).
- **Advice**: The action taken by an aspect at a join point.
- **Pointcut**: Defines where the aspect is applied.
- **Weaving**: Linking aspects with target objects at specified join points.

10. **Explain the types of advice in Spring AOP.**

Answer: Types of advice in Spring AOP:

- **@Before**: Executes before a method.
- **@After**: Executes after a method, regardless of its outcome.
- **@AfterReturning**: Executes after a method returns successfully.
- **@AfterThrowing**: Executes after a method throws an exception.
- **@Around**: Executes before and after a method, providing full control.

11. **How is @Aspect used in Spring?**

Answer: @Aspect is used to define a class as an aspect containing advice methods. It is combined with advice annotations (@Before, @After, etc.) to specify where the advice applies.

Example:

java

```
@Aspect
@Component
public class LoggingAspect {
  @Before("execution(* com.example.service.*.*(..))")
  public void logBeforeMethod() {
    System.out.println("Executing method...");
  }
}
```

12. **What is a pointcut, and how is it used?**

Answer: A pointcut defines specific join points where an advice should be applied. In Spring, pointcuts are expressed as expressions matching methods, classes, or packages.

13. **What are some common use cases for AOP in Spring?**

Answer: Common use cases include:

- **Logging**: Adding logging before/after method execution.
- **Security**: Enforcing security checks.
- **Transaction Management**: Managing transactions.
- **Caching**: Implementing caching mechanisms.

14. **How do you enable AOP in a Spring application?**

Answer: AOP is enabled with @EnableAspectJAutoProxy in a configuration class. This annotation enables support for processing @Aspect annotations.

15. **What is the execution expression in AOP?**

Answer: execution is a pointcut expression used to match method executions in classes. For example, execution(* com.example.service.*.*(..)) matches all methods in classes under the com.example.service package.

Spring MVC (Model-View-Controller)

16. **What is Spring MVC? Describe its main components.**

Answer: Spring MVC (Model-View-Controller) is a web framework in Spring for building web applications and REST APIs. Its components include:

- **@Controller**: Defines a controller for handling web requests.
- **@RequestMapping**: Maps URLs to methods in controllers.
- **Model**: Holds data to be displayed in the view.
- **View**: Represents the user interface, often defined with templates (e.g., JSP, Thymeleaf).

Example:

java

```
@Controller
public class UserController {
  @RequestMapping("/welcome")
  public String welcome(Model model) {
    model.addAttribute("message", "Hello, Spring MVC!");
    return "welcome"; // Refers to welcome.jsp or other view template
  }
}
```

17. **How do you create a REST API using Spring MVC?**

Answer: In Spring MVC, REST APIs are created using @RestController, @GetMapping, @PostMapping, etc., to map HTTP requests to Java methods.

Example:

java

```
@RestController
```

```
@RequestMapping("/api/users")
public class UserController {
   @GetMapping("/{id}")
   public User getUserById(@PathVariable Long id) {
      return userService.findUserById(id);
   }

   @PostMapping
   public User createUser(@RequestBody User user) {
      return userService.saveUser(user);
   }
}
```

18. **What is @RequestMapping, and how does it work?**

Answer: @RequestMapping maps HTTP requests to handler methods of MVC controllers, supporting various attributes such as value, method, params, headers.

Example:

java

```
@Controller
public class ProductController {
   @RequestMapping(value = "/product", method = RequestMethod.GET)
   public String showProductPage() {
      return "product";
   }
}
```

19. **Explain the difference between @Controller and @RestController.**

Answer: @Controller is used to define a controller in a web application that returns views (like JSP, Thymeleaf), while @RestController is a specialization for REST APIs, returning JSON/XML responses directly.

20. **How do you handle exceptions in Spring MVC?**

Answer: Exceptions in Spring MVC can be handled using @ExceptionHandler at the controller level or @ControllerAdvice globally.

Example:

java

```
@ControllerAdvice
public class GlobalExceptionHandler {
   @ExceptionHandler(UserNotFoundException.class)
   public ResponseEntity<String> handleUserNotFound(UserNotFoundException ex) {
      return new ResponseEntity<>(ex.getMessage(), HttpStatus.NOT_FOUND);
   }
}
```

Spring Boot

21. **What is Spring Boot, and what problem does it solve?**

Answer: Spring Boot is a framework built on top of Spring to simplify application development by providing embedded servers, auto-configuration, and opinionated defaults. It reduces the need for extensive XML configuration, allowing developers to start new projects quickly.

22. **Explain @SpringBootApplication and its components.**

Answer: @SpringBootApplication is a combination of:

- @Configuration: Indicates a configuration class.
- @EnableAutoConfiguration: Enables auto-configuration.
- @ComponentScan: Scans for components within the package.

Example:

java

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

23. **How does Spring Boot handle application configuration with application.properties or application.yml?**

Answer: application.properties or application.yml is used to configure application-specific properties, such as server port, database settings, and custom configurations.

24. **What are Spring Boot profiles, and how are they used?**

Answer: Profiles in Spring Boot allow developers to separate configurations for different environments (e.g., dev, prod). Profiles can be activated in application.properties using spring.profiles.active=dev.

25. **What is Spring Boot Actuator, and what are some of its uses?**

Answer: Actuator in Spring Boot provides production-ready features like monitoring and management through various endpoints (/health, /metrics). It enables administrators to monitor application performance.

Spring Data JPA

26. **What is Spring Data JPA?**

Answer: Spring Data JPA is a part of the Spring Data project that simplifies JPA-based data access by providing CRUD operations, custom query methods, and repository interfaces (CrudRepository, JpaRepository).

27. **Explain the role of @Entity, @Id, and @GeneratedValue annotations in JPA.**

Answer: @Entity marks a class as a JPA entity. @Id denotes the primary key, and @GeneratedValue specifies how the ID should be generated.

Example:

java

```
@Entity
public class User {
  @Id
  @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id;
  private String name;
}
```

28. **How do you define custom query methods in Spring Data JPA?**

Answer: Custom query methods can be defined by naming conventions in the repository interface (e.g., findByName) or by using the @Query annotation.

Spring Transactions

29. **What is @Transactional in Spring, and what does it do?**

Answer: @Transactional manages transactions automatically by wrapping methods in a transaction. If an exception occurs, it rolls back; otherwise, it commits.

30. **Explain transaction isolation levels and their types in Spring.**

Answer: Isolation levels control how transactions interact with each other. Types include:

- **READ_UNCOMMITTED**: Allows dirty reads.
- **READ_COMMITTED**: Prevents dirty reads.
- **REPEATABLE_READ**: Prevents non-repeatable reads.
- **SERIALIZABLE**: Prevents phantom reads.

31. **What are transaction propagation levels in Spring?**

Answer: Propagation levels determine how transactions work with nested transactions. Examples:

- **REQUIRED**: Uses an existing transaction or creates a new one.
- **REQUIRES_NEW**: Always creates a new transaction.

Spring Security

32. **What is Spring Security, and why is it used?**

Answer: Spring Security provides authentication, authorization, and other security features for Java applications, making it easier to secure web applications and REST APIs.

33. **How is authentication and authorization implemented in Spring Security?**

Answer: Authentication is validating user identity, and authorization checks user permissions. Spring Security handles this using filters, configuration classes, and @EnableWebSecurity.

34. **Explain method-level security in Spring with @PreAuthorize and @PostAuthorize.**

Answer: @PreAuthorize checks access before entering a method, while @PostAuthorize checks access after execution.

35. **What is CSRF, and how does Spring Security handle it?**

Answer: Cross-Site Request Forgery (CSRF) is an attack that forces users to execute unwanted actions. Spring Security enables CSRF protection by default for web applications.

36. **Explain OAuth2 and JWT with an example in Spring Security.**

Answer: OAuth2 is an authorization framework, and JWT is a token format. In Spring Security, JWT tokens can be generated for users and validated for secure RESTful services.

Spring Cloud

37. **What is Spring Cloud, and how is it used in microservices?**

Answer: Spring Cloud provides tools for building microservices, including service discovery, load balancing, and configuration management. It simplifies distributed systems' complexity.

38. **Explain service discovery with Eureka in Spring Cloud.**

Answer: Eureka, part of Spring Cloud Netflix, is used for service registration and discovery, allowing microservices to locate each other without hard-coding IP addresses.

Spring Cloud

39. **What is client-side load balancing in Spring Cloud, and how does Ribbon work?**

Answer: Client-side load balancing means that each service instance (usually a microservice) has the intelligence to decide which instance to call. **Ribbon** is a client-side load-balancer in Spring Cloud that distributes requests across available instances of a service. When used with Eureka, Ribbon dynamically finds available instances and balances load across them.

Example:

java

```
@LoadBalanced

@Bean

public RestTemplate restTemplate() {

   return new RestTemplate();

}
```

40. **How does Spring Cloud Config work for centralized configuration management?**

Answer: Spring Cloud Config provides server and client-side support for centralized configuration, allowing apps to fetch configurations from a centralized server. The **Config Server** retrieves configuration properties from a Git repository and serves them to client applications, enabling consistent configuration management across environments.

41. **What is Hystrix, and how does it implement circuit breaking in Spring Cloud?**

Answer: **Hystrix** is a fault-tolerance library in Spring Cloud Netflix that enables **circuit breaking** to prevent cascading failures in microservices. When a service call fails repeatedly, Hystrix "opens" the circuit to stop calls temporarily, avoiding overloading failing services.

Example:

java

```
@HystrixCommand(fallbackMethod = "fallback")

public String callExternalService() {
    // service call
}


public String fallback() {
    return "Service temporarily unavailable";
}
```

42. **What is a Spring Cloud Gateway, and how is it used in microservices?**

Answer: Spring Cloud Gateway is a library that provides a simple way to route requests to microservices, acting as an API gateway. It handles cross-cutting concerns like authentication, logging, and rate limiting, making it ideal for handling complex routing needs.

Example:

yaml

```
spring:
  cloud:
    gateway:
      routes:
        - id: user-service
          uri: lb://USER-SERVICE
          predicates:
            - Path=/user/**
```

43. **How does Spring Cloud Stream enable messaging between microservices?**

Answer: Spring Cloud Stream is a framework that allows building message-driven microservices. It abstracts messaging brokers like Kafka and RabbitMQ, allowing microservices to communicate asynchronously using message channels defined in code.

Example:

java

```
@EnableBinding(Source.class)
public class MessagePublisher {
    @Autowired
    private MessageChannel output;
    public void sendMessage(String message) {
        output.send(MessageBuilder.withPayload(message).build());
    }
}
```

Spring Security

44. **What is OAuth2 in Spring Security, and how is it implemented?**

Answer: **OAuth2** is an authorization framework that enables secure access to resources without sharing credentials. Spring Security OAuth2 allows clients to receive an access token from an authorization server, which they can use to authenticate against protected APIs.

Example:

java

```
@Configuration
@EnableAuthorizationServer
public class AuthorizationServerConfig extends AuthorizationServerConfigurerAdapter {
    // Configure clients, token store, etc.
}
```

45. **How can you implement JWT-based authentication in Spring Security?**

Answer: JWT (JSON Web Token) is a token format used to securely transmit information between parties. Spring Security can be configured to issue JWT tokens upon successful login and validate them for each request.

Example:

java

```
@PostMapping("/login")
public ResponseEntity<?> login(@RequestBody AuthRequest authRequest) {
    // Authenticate user and generate JWT
    String token = jwtService.generateToken(authRequest.getUsername());
    return ResponseEntity.ok(new AuthResponse(token));
}
```

46. **Explain the purpose of @EnableGlobalMethodSecurity.**

Answer: @EnableGlobalMethodSecurity enables method-level security annotations like @PreAuthorize and @PostAuthorize, allowing granular control over security at the method level.

Spring Transactions

47. **How do transaction rollbacks work in Spring?**

Answer: Spring's transaction management automatically rolls back a transaction if an exception is thrown from a method annotated with @Transactional, specifically if it's a **runtime exception**. Checked exceptions by default do not trigger rollbacks unless specified.

Example:

java

```
@Transactional(rollbackFor = Exception.class)

public void updateData() {

   // Method body

}
```

48. **What is Propagation.REQUIRES_NEW, and when is it used?**

Answer: Propagation.REQUIRES_NEW starts a new transaction, suspending any existing transaction. This is useful when a particular method's transaction should not be affected by the outer transaction's outcome.

Example:

java

```
@Transactional(propagation = Propagation.REQUIRES_NEW)

public void logAction(String action) {

   // Independent transaction for logging

}
```

Spring Data JPA

49. **Explain the purpose of @Query annotation in Spring Data JPA.**

Answer: @Query allows defining custom queries directly in the repository interface using JPQL or native SQL, offering more control than query methods.

Example:

java

```
@Query("SELECT u FROM User u WHERE u.email = ?1")

User findByEmail(String email);
```

50. **What is @Modifying, and when would you use it in Spring Data JPA?**

Answer: @Modifying is used with @Query for **update** or **delete** operations. It indicates that the query will modify data rather than just fetching it.

Example:

java

```
@Modifying

@Query("UPDATE User u SET u.status = ?1 WHERE u.id = ?2")

void updateUserStatus(String status, Long id);
```

Additional Questions

51. **How do you handle lazy loading exceptions in Spring Data JPA?**

Answer: Lazy loading exceptions occur when entities with lazy-loaded fields are accessed outside of a transaction. To handle this, ensure queries are run within the transaction scope, or use **JOIN FETCH** in the query to pre-load data.

52. **What is the difference between @EnableJpaRepositories and @RepositoryScan in Spring?**

Answer: @EnableJpaRepositories is used to scan for Spring Data JPA repositories, whereas @RepositoryScan is used to scan for custom repository components.

53. **How does Spring Boot simplify dependency management?**

Answer: Spring Boot provides a **dependency management plugin** in spring-boot-starter which ensures compatible versions of libraries, minimizing conflicts and version incompatibility issues.

54. **What are Spring Boot DevTools, and how do they aid development?**

Answer: Spring Boot DevTools enables **live reload**, **automatic restart**, and **enhanced logging** for faster development, updating applications upon saving changes without a full rebuild.

55. **Explain @ComponentScan and its role in Spring.**

Answer: @ComponentScan directs Spring to scan for annotated components (@Component, @Service, etc.) within a specified package, ensuring dependency injection and bean creation.

56. **What is the role of ApplicationContext in Spring?**

Answer: ApplicationContext is a Spring container that initializes and manages beans, providing dependency injection, event handling, and bean lifecycle management.

57. **What is @Value annotation in Spring?**

Answer: @Value is used to inject values from properties files or environment variables directly into Spring-managed beans.

Example:

java

```
@Value("${app.name}")

private String appName;
```

58. **How does Spring MVC handle file uploads?**

Answer: Spring MVC handles file uploads with MultipartFile and configurations in multipartResolver, allowing files to be received and processed within controllers.

59. **What is DispatcherServlet, and how does it work in Spring MVC?**

Answer: DispatcherServlet is the front controller in Spring MVC, routing requests to appropriate handlers based on request mappings.

60. **How do you use @RestControllerAdvice for centralized error handling?**

Answer: @RestControllerAdvice manages global exceptions for REST APIs, enabling custom handling for specific exceptions, returning standardized error responses.