Top 50 Most Challenging Spring Interview Questions

PART - 1

LinkedIn
https://www.linkedin.com/in/venkattramana

1. What is the difference between `@Transactional(propagation = Propagation.REQUIRES_NEW)` and `Propagation.NESTED` in Spring?

Answer:

- `REQUIRES_NEW`: Suspends the current transaction and creates a **new, independent** transaction.
- `NESTED`: Runs in a **nested transaction** within the current one; rollback only affects the nested block.

Example:

```
@Transactional
public void outerMethod() {
    service.methodWithNested(); // rollback only inside this if nested fails
    service.methodWithRequiresNew(); // runs in separate tx
}

@Transactional(propagation = Propagation.NESTED)
public void methodWithNested() {
    // Fails here → rollback only this nested part
    throw new RuntimeException("Nested fails");
}

@Transactional(propagation = Propagation.REQUIRES_NEW)
public void methodWithRequiresNew() {
    // Completely new transaction
}
```

---

2. How does Spring handle circular dependencies, and how can you resolve them?

Answer:
Spring resolves circular dependencies via **singleton bean injection through setter injection**, not constructor injection.

Example:

```
@Component
public class A {
    private B b;

    @Autowired
    public void setB(B b) { this.b = b; }
}

@Component
public class B {
    private A a;

    @Autowired
    public void setA(A a) { this.a = a; }
}
```

To resolve constructor-based circular dependencies: Use `@Lazy`, redesign, or inject via setters.

---

3. What happens if a Spring bean is `@Transactional`, but the method is called from inside the same class?

Answer:
Self-invocation bypasses the proxy, so the transaction is ignored.

Fix: Move the method to a separate bean or call via `ApplicationContext.getBean(...)`.

Example:

```
@Component
public class OrderService {
    @Transactional
    public void placeOrder() {
        validatePayment(); // Not transactional!
    }

    @Transactional
    public void validatePayment() {
        // Transaction not triggered here
    }
}
```

---

4. Explain how `@EventListener` works in Spring. How can you handle async events?

Answer:

`@EventListener` listens for published Spring events.

Use `@Async` + `@EnableAsync` to make them asynchronous.

Example:

```
@Component
public class NotificationListener {
    @Async
    @EventListener
    public void onOrderPlaced(OrderEvent event) {
        // Send email/SMS notification
    }
}

@Configuration
@EnableAsync
class AsyncConfig {}
```

---

5. How does Spring Boot auto-configuration work internally?

Answer:
Spring Boot uses `spring.factories` under the hood:

- It scans for `@Configuration` classes listed in `META-INF/spring.factories`.

Example:

```
# META-INF/spring.factories
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.example.MyCustomAutoConfig
```

```
@Configuration
public class MyCustomAutoConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

---

6. What are the differences between `@Component`, `@Bean`, and `@Configuration`?

Answer:

Annotation	Use Case	Managed by Spring?
`@Component`	On class level for auto-scan	Yes
`@Bean`	On method level inside `@Config`	Yes
`@Configuration`	Declares configuration class	Yes (CGLIB proxy)

Example:

```
@Component
public class ComponentBean {}

@Configuration
public class ConfigClass {
    @Bean
    public BeanObject myBean() {
        return new BeanObject();
    }
}
```

---

7. How does `@RequestScope`, `@SessionScope`, and `@ApplicationScope` differ?

Answer:

- `@RequestScope`: Bean lives per HTTP request.
- `@SessionScope`: Lives per session.
- `@ApplicationScope`: Lives for the entire app.

Example:

```
@Component
@RequestScope
public class RequestScopedBean {}

@Component
@SessionScope
public class SessionScopedBean {}
```

---

8. Explain method-level security using `@PreAuthorize`, `@PostAuthorize`. How are they evaluated?

Answer:

- `@PreAuthorize`: Checks **before** method executes.

- `@PostAuthorize`: Checks **after** method executes (can use return object).

Example:

```
@PreAuthorize("hasRole('ADMIN')")
public void deleteUser(int id) {}

@PostAuthorize("returnObject.owner == authentication.name")
public User getUserDetails(int id) {
    return userRepo.findById(id);
}
```

---

9. What is Spring's `BeanPostProcessor`, and where is it useful?

Answer:
It allows **custom logic before/after bean initialization**.

Example:

```
@Component
public class CustomBeanPostProcessor implements BeanPostProcessor {
    public Object postProcessBeforeInitialization(Object bean, String name) {
        System.out.println("Before Init: " + name);
        return bean;
    }
}
```

Used in AOP, lifecycle hooks, and proxy enhancement.

---

10. How can you implement dynamic multi-tenancy in Spring Boot + Hibernate?

Answer:
Use `MultiTenantConnectionProvider` and `CurrentTenantIdentifierResolver`.

Example:

```
@Component
public class CurrentTenantResolver implements CurrentTenantIdentifierResolver {
    public String resolveCurrentTenantIdentifier() {
        return TenantContext.getCurrentTenant();
    }

    public boolean validateExistingCurrentSessions() {
        return true;
    }
}
```

```
@Bean
public LocalContainerEntityManagerFactoryBean entityManagerFactory(...) {
    properties.put(Environment.MULTI_TENANT, MultiTenancyStrategy.SCHEMA);
    properties.put(Environment.MULTI_TENANT_IDENTIFIER_RESOLVER, tenantResolver);
    // ...
}
```

---

11. What is the use of `@ControllerAdvice` and how is it different from `@ExceptionHandler` in controllers?

Answer:

- `@ControllerAdvice` is a global exception handler across multiple controllers.
- `@ExceptionHandler` is specific to one controller.

Example:

```
@ControllerAdvice
public class GlobalExceptionHandler {
    @ExceptionHandler(ResourceNotFoundException.class)
    public ResponseEntity<?> handleNotFound(ResourceNotFoundException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(ex.getMessage());
    }
}
```

---

12. How does Spring Boot support graceful shutdown, and how do you implement it?

Answer:
Spring Boot supports graceful shutdown by delaying shutdown until current requests complete.

To enable:

```yaml
# application.yml
server:
  shutdown: graceful
spring:
  lifecycle:
    timeout-per-shutdown-phase: 30s
```

Custom hook:

```java
@Component
public class MyShutdownHook implements DisposableBean {
    public void destroy() {
        // Cleanup tasks
        System.out.println("Application shutting down...");
    }
}
```

---

13. How to create a custom Spring Boot Starter?

Answer:

1. Create a new Maven module.
2. Add `spring.factories`.
3. Provide default auto-config classes.

Example:

```java
@Configuration
public class MyLibraryAutoConfig {
    @Bean
    public MyService myService() {
        return new MyService();
    }
}
```

```
# META-INF/spring.factories
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
com.example.MyLibraryAutoConfig
```

---

14. How does Spring Security's FilterChainProxy work under the hood?

Answer:
`FilterChainProxy` holds multiple `SecurityFilterChain` objects mapped to different URL patterns.

Each chain contains filters like:

- `UsernamePasswordAuthenticationFilter`
- `BasicAuthenticationFilter`
- `ExceptionTranslationFilter`

Example:

```
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable()
        .authorizeHttpRequests()
        .requestMatchers("/admin/**").hasRole("ADMIN")
        .anyRequest().authenticated();
    return http.build();
}
```

---

15. What is the role of `@EnableConfigurationProperties` and how does it work?

Answer:
It binds external configuration (YAML/properties) to POJOs.

Example:

```
my:
  app:
    name: ChatGPT
    version: 1.0
```

```java
@Component
@ConfigurationProperties(prefix = "my.app")
public class AppConfig {
    private String name;
    private String version;
    // Getters and Setters
}

@Configuration
@EnableConfigurationProperties(AppConfig.class)
public class AppConfigEnabler {}
```

---

16. Explain Spring Retry mechanism and its configuration.

Answer:
Spring Retry allows methods to be automatically retried on failure.

Add dependency:

```xml
<dependency>
  <groupId>org.springframework.retry</groupId>
  <artifactId>spring-retry</artifactId>
</dependency>
```

Usage:

```java
@EnableRetry
@Configuration
public class RetryConfig {}

@Component
public class RetryService {
    @Retryable(maxAttempts = 3, value = {IOException.class})
    public void unstableCall() throws IOException {
        // retry on IOException
    }

    @Recover
    public void recover(IOException e) {
        // fallback logic
    }
}
```

17. How to implement request rate limiting in Spring Boot?

Answer:
Use libraries like Bucket4j or Redis.

Example with Bucket4j (in-memory):

```
@Component
public class RateLimiterFilter extends OncePerRequestFilter {
    private final Bucket bucket = Bucket4j.builder()
        .addLimit(Bandwidth.simple(5, Duration.ofMinutes(1)))
        .build();

    protected void doFilterInternal(...) {
        if (bucket.tryConsume(1)) {
            filterChain.doFilter(request, response);
        } else {
            response.setStatus(HttpStatus.TOO_MANY_REQUESTS.value());
        }
    }
}
```

---

18. How to configure distributed tracing in Spring Boot microservices (Zipkin)?

Answer:
Add `spring-cloud-starter-zipkin` + properties.

Example:

```
spring:
  zipkin:
    base-url: http://localhost:9411
    sender:
      type: web
  sleuth:
    sampler:
      probability: 1.0
```

Add Dependency:

```
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-zipkin</artifactId>
</dependency>
```

---

19. What is the use of `@Profile` and how to load beans conditionally based on environment?

Answer:
`@Profile` defines beans conditionally based on active Spring profiles.

Example:

```
@Profile("dev")
@Bean
public DataSource devDataSource() {
    return new H2DataSource();
}

@Profile("prod")
@Bean
public DataSource prodDataSource() {
    return new OracleDataSource();
}
```

Set profile using:

-Dspring.profiles.active=dev

---

20. Explain the role of `WebMvcConfigurer` and how to customize Spring MVC configuration.

Answer:
`WebMvcConfigurer` provides hook methods for:

- CORS configuration
- Interceptors
- Formatters

- Resource handling

Example:

```
@Configuration
public class WebConfig implements WebMvcConfigurer {
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/api/**").allowedOrigins("*");
    }

    public void addInterceptors(InterceptorRegistry registry) {
        registry.addInterceptor(new LoggingInterceptor());
    }
}
```

---

21. How do you secure inter-service communication in Spring Boot microservices?

Answer:
You can secure communication using:

- **OAuth2** with JWT tokens
- **Mutual TLS**
- **API Gateway Authentication**

Example (OAuth2 with Feign):

```
security:
  oauth2:
    client:
      registration:
        my-client:
          client-id: client-id
          client-secret: secret
```

Feign Client with Token Relay:

```
@FeignClient(name = "orders", configuration = OAuth2FeignConfig.class)
public interface OrderClient {
    @GetMapping("/api/orders")
    List<Order> getOrders();
}
```

---

22. How does Spring Cloud Config work and how does it refresh properties at runtime?

Answer:
Spring Cloud Config Server fetches properties from Git or Vault and injects them into clients. Use `@RefreshScope` + actuator to refresh values.

Example:

```
@RefreshScope
@RestController
public class MessageController {
    @Value("${message}")
    private String message;

    @GetMapping("/msg")
    public String getMessage() {
        return message;
    }
}
```

Trigger refresh:

curl -X POST http://localhost:8080/actuator/refresh

---

23. What is Circuit Breaker in Spring Cloud and how do you implement it with Resilience4j?

Answer:
A circuit breaker prevents system overload by stopping repeated failed requests.

Example:

```
@CircuitBreaker(name = "inventoryService", fallbackMethod = "fallbackInventory")
public String callInventory() {
    return restTemplate.getForObject("http://inventory/api", String.class);
}

public String fallbackInventory(Exception e) {
    return "Inventory unavailable";
}
```

Config in `application.yml`:

```yaml
resilience4j.circuitbreaker:
  instances:
    inventoryService:
      registerHealthIndicator: true
      slidingWindowSize: 10
      failureRateThreshold: 50
```

---

24. How to implement distributed locking in Spring (e.g., for scheduled tasks)?

Answer:
Use Redis-backed locking using libraries like **ShedLock**.

Example:

```java
@Scheduled(cron = "0 * * * * *")
@SchedulerLock(name = "scheduledTask", lockAtLeastFor = "10s")
public void scheduledTask() {
    // Only one instance runs this
}
```

Dependency:

```xml
<dependency>
  <groupId>net.javacrumbs.shedlock</groupId>
  <artifactId>shedlock-spring</artifactId>
</dependency>
```

---

25. What is the difference between `@RequestBody` and `@ModelAttribute`?

Answer:

Annotation	Parses From	Use Case
`@RequestBody`	JSON/XML body	REST API
`@ModelAttribute`	Form data / Query Params	Web forms (MVC)

Example:

```
@PostMapping("/create")
public ResponseEntity<?> createUser(@RequestBody User user) {...}

@GetMapping("/form")
public String submitForm(@ModelAttribute User user) {...}
```

---

26. Explain how `@Aspect`, `JoinPoint`, and `Pointcut` work together in Spring AOP.

Answer:

- `@Aspect`: Declares a class as an aspect.
- `@Pointcut`: Defines a matching expression.
- `@Before`, `@After`: Advice methods that run on matched points.

Example:

```
@Aspect
@Component
public class LoggingAspect {
    @Pointcut("execution(* com.app.service.*.*(..))")
    public void serviceMethods() {}

    @Before("serviceMethods()")
    public void logBefore(JoinPoint jp) {
        System.out.println("Calling: " + jp.getSignature());
    }
}
```

---

27. How does Spring Boot DevTools work under the hood for live reload and restarts?

Answer:
Spring DevTools uses a separate classloader to watch non-static classes and triggers a **context restart** (not full JVM restart) on file change.

Features:

- Auto-reload via `RestartClassLoader`
- LiveReload integration with browsers

---

28. How do you implement content negotiation in Spring Boot (XML/JSON responses)?

Answer:
Configure `HttpMessageConverters` or let Spring auto-detect based on headers.

Example:

```
@GetMapping(value = "/data", produces = {MediaType.APPLICATION_JSON_VALUE,
                                         MediaType.APPLICATION_XML_VALUE})
public MyData getData() {
    return new MyData("Venkat", 42);
}
```

Spring chooses response type based on:

- `Accept` header
- URL extension (if enabled)

---

29. How does Spring Boot Actuator monitor application health?

Answer:
Actuator exposes `/actuator/health`, `/metrics`, `/env` endpoints.

Example:

```
management:
  endpoints:
    web:
      exposure:
        include: health,metrics,env
```

Custom health indicator:

```
@Component
public class DBHealthIndicator implements HealthIndicator {
    public Health health() {
        boolean dbUp = checkDB();
        return dbUp ? Health.up().build() : Health.down().withDetail("DB", "Down").build();
    }
}
```

---

30. What is the difference between `@ConditionalOnProperty` and `@ConditionalOnMissingBean`?

Answer:

Annotation	Use Case
`@ConditionalOnProperty`	Load bean based on property value
`@ConditionalOnMissingBean`	Load only if bean not defined

Example:

```
@Bean
@ConditionalOnProperty(name = "feature.enable", havingValue = "true")
public MyFeature featureBean() {...}

@Bean
@ConditionalOnMissingBean(MyService.class)
public MyService defaultService() {...}
```

---

31. What is the difference between Mono and Flux in Spring WebFlux?

Answer:

Return Type	Meaning	Use Case

`Mono<T>`	0 or 1 element	REST responses
`Flux<T>`	0 to N elements	Streaming data/events

Example:

```
@GetMapping("/mono")
public Mono<String> getMono() {
    return Mono.just("Hello");
}

@GetMapping("/flux")
public Flux<String> getFlux() {
    return Flux.just("A", "B", "C");
}
```

---

32. How does backpressure work in Spring Reactive streams?

Answer:
Backpressure allows consumers to **signal producers** about how much data they can handle, preventing memory overload.

Spring uses **Project Reactor**, which supports backpressure through `Publisher` and `Subscriber`.

Example:

```
Flux.range(1, 100)
    .onBackpressureDrop()
    .subscribe(System.out::println);
```

---

33. How do you test `@RestController` in Spring Boot using `@WebMvcTest`?

Answer:
Use `@WebMvcTest` for controller-layer testing and mock dependencies.

Example:

```java
@WebMvcTest(UserController.class)
public class UserControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @MockBean
    private UserService userService;

    @Test
    public void testGetUser() throws Exception {
        Mockito.when(userService.getUser(1)).thenReturn(new User(1, "Venkat"));
        mockMvc.perform(get("/users/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.name").value("Venkat"));
    }
}
```

---

34. What is the purpose of `@SpyBean` and `@MockBean` in Spring Boot testing?

Answer:

- `@MockBean`: Replace bean with a Mockito mock.
- `@SpyBean`: Spy on the actual bean to partially mock behavior.

Example:

```java
@MockBean
private PaymentService paymentService; // Replaces original

@SpyBean
private EmailService emailService; // Real bean, with spying
```

---

35. How does Spring Kafka handle message retries and dead-letter topics?

Answer:
Use `DefaultErrorHandler` and configure retries or forward to **dead-letter-topic**.

Example:

```
@Bean
public DefaultErrorHandler errorHandler() {
    return new DefaultErrorHandler(
        new DeadLetterPublishingRecoverer(kafkaTemplate),
        new FixedBackOff(1000L, 3)); // retry 3 times
}
```

```
@KafkaListener(topics = "orders")
public void process(String message) {
    // If fails after retries, goes to DLT
}
```

---

36. How does Spring Security manage session fixation attacks?

Answer:
By default, Spring Security changes the session ID on login to prevent session fixation.

To configure:

```
http
  .sessionManagement()
  .sessionFixation().migrateSession(); // default
```

Options:

- `none()`: Don't change session
- `newSession()`: Create new session without copying attributes

---

37. What is `WebClient` and how does it replace `RestTemplate`?

Answer:
`WebClient` is a **non-blocking, reactive** HTTP client that supports async requests.

Example:

```
WebClient webClient = WebClient.create();

Mono<String> response = webClient.get()
    .uri("http://localhost:8080/data")
    .retrieve()
    .bodyToMono(String.class);
```

---

38. How do you perform method-level validation using Spring Boot and Hibernate Validator?

Answer:

Use `@Validated` at class level and `@Min`, `@NotNull`, etc., on method parameters.

Example:

```
@Service
@Validated
public class OrderService {
    public void placeOrder(@NotNull @Min(1) Integer orderId) {
        // Validation auto-triggered
    }
}
```

---

39. What is Spring Cloud Gateway and how does it differ from Zuul?

Answer:

Feature	Spring Cloud Gateway	Zuul 1
Reactive support	Yes (Project Reactor)	No
Performance	High (Netty)	Lower (Servlet-based)
Path Rewriting, Filters	Declarative + Code	Mostly code

Example:

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id: user_route
          uri: http://user-service
          predicates:
            - Path=/users/**
```

---

40. How can you create a custom Spring Boot actuator endpoint?

Answer:
Use `@Endpoint` and expose it via actuator configuration.

Example:

```java
@Endpoint(id = "customhealth")
@Component
public class CustomHealthEndpoint {
    @ReadOperation
    public String health() {
        return "OK";
    }
}
```

Enable in `application.yml`:

```yaml
management:
  endpoints:
    web:
      exposure:
        include: "*"
```

---

41. How do you implement request tracing and correlation ID in Spring Boot microservices?

Answer:
Assign a unique ID per request (correlation ID) and pass it across service calls.

Example using Filter:

```
@Component
public class CorrelationIdFilter extends OncePerRequestFilter {
    public static final String CORRELATION_ID = "X-Correlation-Id";

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
            throws ServletException, IOException {
        String correlationId = request.getHeader(CORRELATION_ID);
        if (correlationId == null) {
            correlationId = UUID.randomUUID().toString();
        }
        MDC.put(CORRELATION_ID, correlationId);
        response.setHeader(CORRELATION_ID, correlationId);
        chain.doFilter(request, response);
        MDC.remove(CORRELATION_ID);
    }
}
```

---

42. What is functional bean registration in Spring Boot and why use it?

Answer:
Functional bean registration provides a **Java-config-only** way to register beans without annotations, improving clarity and startup time.

Example:

```
public class App {
    public static void main(String[] args) {
        ApplicationContext context = new GenericApplicationContext();
        ((GenericApplicationContext) context).registerBean(MyService.class);
    }
}
```

---

43. How to expose Prometheus metrics with Spring Boot and visualize in Grafana?

Answer:

1. Add dependency:

```xml
<dependency>
  <groupId>io.micrometer</groupId>
  <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

2. Enable endpoint:

```yaml
management:
  endpoints:
    web:
      exposure:
        include: prometheus
```

3. Access metrics at: `http://localhost:8080/actuator/prometheus`

---

44. How can you build a modular Spring Boot application using Java modules (JPMS)?

Answer:
Split app into multiple modules with `module-info.java`.

Example:

```
module user.service {
    requires spring.context;
    exports com.example.user;
}
```

Register `module-path` in build tools like Maven/Gradle, and avoid reflection-related features incompatible with JPMS.

---

45. How do you secure REST APIs using API key headers in Spring Boot?

Answer:
Use a `OncePerRequestFilter` to intercept and validate a custom API key header.

Example:

```java
@Component
public class ApiKeyFilter extends OncePerRequestFilter {
    @Value("${api.key}")
    private String key;

    protected void doFilterInternal(HttpServletRequest req, HttpServletResponse res, FilterChain chain) throws ... {
        String header = req.getHeader("X-API-KEY");
        if (!key.equals(header)) {
            res.setStatus(HttpStatus.UNAUTHORIZED.value());
            return;
        }
        chain.doFilter(req, res);
    }
}
```

---

46. What is the purpose of `@Conditional` and how do you create a custom condition?

Answer:
`@Conditional` allows beans to be loaded based on custom logic.

Custom Condition:

```java
public class OnDevCondition implements Condition {
    public boolean matches(ConditionContext context, AnnotatedTypeMetadata metadata) {
        return "dev".equals(context.getEnvironment().getProperty("env"));
    }
}
```

Usage:

```java
@Bean
@Conditional(OnDevCondition.class)
public MyService devService() {
    return new DevService();
}
```

---

47. How do you implement canary deployment with Spring Boot + Kubernetes?

Answer:
Create two deployments:

- **Stable version (v1)** and
- **Canary version (v2)**

Use **K8s Ingress or Istio** to split traffic (e.g., 90%-10%).

Istio VirtualService

Example:

```
http:
  - route:
      - destination:
          host: my-service
          subset: stable
        weight: 90
      - destination:
          host: my-service
          subset: canary
        weight: 10
```

---

48. How do you manage secrets in Spring Boot securely?

Answer:
Best practices:

- Avoid plaintext `.properties`
- Use Spring Cloud Vault, AWS Secrets Manager, or Azure Key Vault.

Vault Example:

```
spring:
  cloud:
    vault:
      uri: http://localhost:8200
      authentication: token
      token: s.xxxxxxxx
```

Inject values:

```
@Value("${db.password}")
private String password;
```

---

49. How to handle JSON Patch or Merge Patch in Spring Boot?

Answer:
Use Jackson and `JsonMergePatch` or libraries like **json-patch**.

Example:

```
@PatchMapping("/user/{id}")
public ResponseEntity<User> patchUser(@PathVariable Long id, @RequestBody JsonMergePatch patch) {
    User original = userRepo.findById(id);
    User patched = applyPatch(patch, original, User.class);
    return ResponseEntity.ok(userRepo.save(patched));
}
```

---

50. How do you implement multi-tenancy in Spring Security?

Answer:
Approaches:

1. Tenant from JWT
2. Tenant from subdomain/header
3. Tenant stored in `SecurityContext`

Example:

```
public class TenantFilter extends OncePerRequestFilter {
    protected void doFilterInternal(...) {
        String tenantId = request.getHeader("X-Tenant-ID");
        TenantContext.setTenant(tenantId); // ThreadLocal
        chain.doFilter(request, response);
        TenantContext.clear();
    }
}
```