**Frequently Asked Questions in Spring Boot**

**1. Calling a REST Service from Another REST Service**

To call a REST service from another REST service in a Spring Boot application:

* Use **RestTemplate** or **WebClient** (for non-blocking calls).
* Create a bean of RestTemplate or WebClient in your configuration class.
* Inject the bean into your service class and use it to make HTTP calls.

**Example with RestTemplate:**

@RestController

public class MyController {

@Autowired

private RestTemplate restTemplate;

@GetMapping("/callService")

public String callAnotherService() {

String url = "http://example.com/api/service";

return restTemplate.getForObject(url, String.class);

}

}

**For multiple REST services:** You can use the same instance of RestTemplate or WebClient. Configure different base URLs and endpoints dynamically.

**2. Converting String to Integer from REST Template Call**

To automate the conversion without manual casting:

* Use **ObjectMapper** from Jackson for JSON responses.
* For XML responses, use a library like JAXB.

**Example with ObjectMapper:**

String response = restTemplate.getForObject("http://example.com/api/service", String.class);

Integer result = new ObjectMapper().readValue(response, Integer.class);

Alternatively, define the expected response type in your REST call:

ResponseEntity<Integer> response = restTemplate.exchange(url, HttpMethod.GET, null, Integer.class);

Integer value = response.getBody();

**3. Bean Injection in Spring**

Beans are injected using:

1. **Field Injection** (@Autowired on fields).
2. **Setter Injection** (@Autowired on setters).
3. **Constructor Injection** (recommended for immutability).

**Example: Constructor Injection:**

@Service

public class MyService {

private final MyRepository myRepository;

@Autowired

public MyService(MyRepository myRepository) {

this.myRepository = myRepository;

}

}

**4. Bean Scopes in Spring**

**Types of Bean Scopes:**

1. **Singleton:** Default scope. One instance per Spring container.
   * **Use case:** Stateless services.
2. **Prototype:** New instance for each request.
   * **Use case:** Stateful objects.
3. **Request:** New instance per HTTP request.
   * **Use case:** Web applications with user-specific data.
4. **Session:** New instance per HTTP session.
   * **Use case:** User preferences in a web app.
5. **Application:** One instance per ServletContext.
   * **Use case:** Shared data across sessions.
6. **WebSocket:** One instance per WebSocket.

**5. Spring Profiling**

Spring profiles allow configuring environments (e.g., dev, test, prod).

* Add @Profile("profileName") to beans.
* Use application.properties to activate a profile: spring.profiles.active=dev.

**6. Autowiring in Spring Boot**

Autowiring automatically resolves and injects beans into classes. Types:

1. **No:** Manual wiring.
2. **ByName:** Matches property name with bean name.
3. **ByType:** Matches property type with bean type.
4. **Constructor:** Matches constructor parameters.

**Use case for Constructor Autowiring:** When dependencies are mandatory and should be immutable.

**7. Inversion of Control (IoC)**

IoC shifts the responsibility of object creation and management to a container (Spring IoC Container). The container manages the lifecycle and dependencies of beans.

**8. Securing Spring Boot Applications**

1. **Authentication and Authorization:** Use Spring Security.
2. **Secure endpoints:** Add @PreAuthorize or @RolesAllowed.
3. **Encrypt data:** Use HTTPS, TLS, or secure keys.
4. **CORS policies:** Restrict origins using @CrossOrigin.

**9. HTTP Response Codes**

* **200 (OK):** Request succeeded.
* **400 (Bad Request):** Invalid request syntax.
* **401 (Unauthorized):** Authentication required.
* **403 (Forbidden):** No permission despite authentication.
* **404 (Not Found):** Resource unavailable.
* **500 (Internal Server Error):** Server-side issue.
* **503 (Service Unavailable):** Server overload.

**10. Combining First and Last Name as Primary Key**

Use a **Composite Key** with @EmbeddedId or @IdClass.

**Example with @IdClass:**

@Entity

@IdClass(NameId.class)

public class User {

@Id

private String firstName;

@Id

private String lastName;

}

**11. Transaction Management**

Annotate service methods with @Transactional for atomic operations.

**Example:**

@Service

public class MyService {

@Transactional

public void performTransaction() {

// Business logic

}

}

**12. HTTP vs HTTPS**

* **HTTP:** Unsecured protocol, plain text data transfer.
* **HTTPS:** Secured with SSL/TLS, encrypted communication.