What are the steps to connect an external database like MySQL or Oracle?

To connect an external database like MySQL or Oracle in a Spring Boot application, you can follow these steps

1. **Include Database Driver** Add the corresponding database driver dependency for MySQL or Oracle in your project's `pom.xml` (Maven) or `build.gradle` (Gradle) file.

2. **Configure Connection Properties** In your `application.properties` or `application.yml` file, specify the necessary configuration properties for the database connection. These properties typically include the URL, username, password, and driver class specific to the database you are connecting to.

3. **Use Spring Data JPA (optional)** If you plan to use Spring Data JPA for database operations, include the necessary dependencies and configure the entity classes, repositories, and other JPA-related settings.

4. **Create Data Source Bean** Create a bean that defines the data source configuration for the external database. In this bean, set the required properties such as URL, username, password, and driver class.

5. **Inject Data Source Inject the data source bean** into the appropriate components or repositories that need database access. You can use dependency injection annotations like `@Autowired` or constructor injection to obtain a reference to the data source.

6. **Perform Database Operations** With the data source configured and injected, you can now use JDBC, JPA, or other database frameworks to perform database operations, execute queries, and interact with the external database.

Inversion of Control vs Dependency Injection?

Inversion of Control (IoC) is a broader design principle that aims to invert the control of object creation and management from the application code to a container or framework. It promotes loose coupling and modular design by delegating the responsibility of creating and managing dependencies to an external entity.

Dependency Injection (DI) is a specific implementation of IoC, where dependencies are "injected" into a class rather than being created internally. DI is a way to achieve IoC by allowing

dependencies to be provided from external sources, such as through constructor parameters, setter methods, or interfaces.

What is Bean in Spring Boot?

In Spring Boot, a “bean” is simply a Java object managed by the Spring framework’s Inversion of Control (IoC) container. These beans are created, configured, and managed by the Spring framework, and they typically represent components or services used in your application. Beans can be defined using annotations or XML configuration and are automatically wired together by Spring, allowing for easy dependency injection and modular application design.

Explain about Bean LifeCycle.

The lifecycle of a bean in the Spring framework consists of several key phases:

1. **Instantiation**: This is the creation of a bean instance. It can happen through constructors, factory methods, or other means.

2. **Population of Properties**: Once the bean is created, Spring sets its properties and dependencies.

3. **Bean Post-Processing**: This is an optional step where you can apply custom logic before and after initialization. You can implement interfaces like `BeanPostProcessor` for this.

4. **Initialization**: After properties are set, the bean’s `init` method is called, if defined. This is where you can perform any initialization tasks.

5. **In Use**: The bean is now available for use in your application. Other beans can inject it, and your application can use its functionality.

6. **Destruction**: When the application context is closed or explicitly, the bean’s `destroy` method, if defined, is called to release resources and perform cleanup.

These phases represent the lifecycle of a Spring bean, allowing you to control and customize its behavior at various points, like setting properties, initializing resources, and cleaning up when no longer needed.

Explain about different types of beans.

Different Types of Beans:

1. **Singleton Bean:** A singleton bean is a single instance of a bean per Spring IoC container. It is created once and shared by multiple requests or references within the application context.

2. **Prototype Bean:** A prototype bean is a new instance of a bean created each time it is requested from the Spring IoC container. Each request for a prototype bean results in the creation of a new instance.

3. **Request Bean:** A request bean is tied to the lifecycle of an HTTP request in web applications. It is created and made available for the duration of a single HTTP request and is destroyed afterward.

4. **Session Bean:** A session bean is tied to the lifecycle of an HTTP session in web applications. It is created when a new session is started and remains active until the session is invalidated or expired.

5. **Application Bean:** An application bean is tied to the lifecycle of a Spring application context. It is created when the application context is initialized and destroyed when the application context is closed.

These are some commonly used types of beans in Spring. Each bean type has its own lifecycle and scope, providing flexibility in managing object instances based on the application's requirements.

What is graceful shutdown in Spring Boot?

Graceful shutdown in Spring Boot refers to the process of gracefully stopping a running Spring Boot application, allowing it to complete its ongoing tasks and clean up resources before shutting down. It ensures that all active requests are processed or interrupted in a controlled manner, preventing abrupt termination.

During a graceful shutdown, Spring Boot initiates a sequence of steps to gracefully stop the application. These steps may include

1. Rejecting New Requests The application stops accepting new requests, ensuring that no new tasks are initiated.

2. Waiting for Active Requests to Complete The application waits for the ongoing requests to finish processing. This allows the application to complete any pending tasks or operations.

3. Shutting Down Components Once all active requests are completed, the application proceeds to shut down its components, releasing resources, closing connections, and performing any necessary cleanup.

4. Notifying External Systems Optionally, the application may notify external systems, such as service registries or load balancers, to mark the application as unavailable during the shutdown process.

Graceful shutdown is particularly important in scenarios where the application needs to perform cleanup tasks, release resources, or ensure data consistency. It helps prevent data loss, incomplete operations, or unexpected behavior during application shutdown.

Spring Boot provides mechanisms and hooks, such as the `ApplicationContext` events, `SmartLifecycle` interface, or custom shutdown hooks, to facilitate graceful shutdown in applications. These mechanisms allow developers to define custom behavior and perform necessary cleanup actions before the application terminates.

What are all the different HTTP Methods?

Here are the commonly used HTTP methods

1. **GET**: Retrieves a resource or data from a server.

2. **POST**: Submits data to be processed to a server, typically resulting in the creation of a new resource.

3. **PUT**: Updates or replaces an existing resource with new data.

4. **DELETE**: Deletes a specified resource.

5. **PATCH**: Partially updates an existing resource with new data.

6. **HEAD**: Retrieves metadata of a resource without fetching the actual content.

7. **OPTIONS**: Returns the allowed HTTP methods and capabilities of a server for a given resource.

These HTTP methods provide different ways to interact with resources on a server, enabling various operations such as retrieving, creating, updating, or deleting data.

What is Idempotency? Explain about Idempotent methods.

In REST (Representational State Transfer), idempotency refers to the property of certain HTTP methods where repeated identical requests have the same effect as a single request. In other words, performing an idempotent operation multiple times produces the same result as if it were done once.

Idempotent methods in REST ensure that even if a request is duplicated or retried due to network issues or client behavior, the system remains in the same state and does not produce unintended side effects. Idempotent methods are designed to be safe for retries without causing inconsistencies or unexpected behavior.

The following HTTP methods are considered idempotent in REST

- GET: Retrieving a resource multiple times does not change the state of the server.

- PUT: Updating a resource with the same data multiple times produces the same result.

- DELETE: Deleting a resource multiple times results in the same outcome.

Designing idempotent methods in RESTful APIs helps ensure reliability, consistency, and fault tolerance in distributed systems, allowing for robust and predictable behavior.

What is @Async in Spring Boot?

The **@Async** annotation in Spring Boot is used to indicate that a method should be executed asynchronously. When a method is annotated with **@Async**, it is executed in a separate thread, allowing the calling thread to continue its execution without waiting for the completion of the annotated method.

By using **@Async**, time-consuming or blocking operations can be offloaded to separate threads, improving the overall responsiveness and performance of the application. This annotation is typically used in scenarios where certain methods can be executed independently and their results are not immediately needed by the calling thread.

To enable asynchronous execution in Spring Boot, you need to configure a task executor bean and annotate the target method with **@Async**. Spring Boot will automatically detect the **@Async** annotation and execute the annotated method asynchronously using the specified task executor.

What is the difference between CrudRepository vs. JPARespository?

The main difference between `CrudRepository` and `JpaRepository` is that `JpaRepository` is an extension of `CrudRepository` that provides additional JPA-specific features.

While both interfaces provide basic CRUD (Create, Read, Update, Delete) operations, `JpaRepository` offers additional functionality such as query creation methods, pagination support, and the ability to flush changes to the database.

In summary, `CrudRepository` is suitable for generic CRUD operations, while `JpaRepository` is more feature-rich and specifically designed for JPA-based applications, providing advanced querying capabilities on top of the basic CRUD operations.

Explain about different starter dependencies of Spring-boot.

Spring Boot starter dependencies are a set of curated dependencies that simplify the configuration and setup of specific functionalities or frameworks within a Spring Boot application. Here are explanations of some commonly used starter dependencies

1. `spring-boot-starter-web` Includes dependencies for building web applications with Spring MVC, including embedded Tomcat or Jetty server, Spring Web, and other related components.

2. `spring-boot-starter-data-jpa` Provides dependencies for working with JPA (Java Persistence API), including Hibernate, Spring Data JPA, and database connectors.

3. `spring-boot-starter-security` Includes dependencies for adding security features to the application, such as authentication, authorization, and encryption. It integrates with Spring Security and offers additional features like OAuth support.

4. `spring-boot-starter-test` Provides dependencies for testing the application, including JUnit, Mockito, and Spring Test. It simplifies the setup of unit tests, integration tests, and other testing scenarios.

5. `spring-boot-starter-cache` Includes dependencies for adding caching capabilities to the application. It integrates with popular caching libraries like Ehcache, Caffeine, or Redis.

6. `spring-boot-starter-amqp` Provides dependencies for working with messaging and AMQP (Advanced Message Queuing Protocol) implementations, such as RabbitMQ.

7. `spring-boot-starter-data-redis` Includes dependencies for using Redis as a data store, providing features for caching, distributed sessions, and more.

These are just a few examples of the various starter dependencies available in Spring Boot. Each starter simplifies the configuration and setup of a specific functionality or integration, allowing developers to quickly include and utilize the required features in their Spring Boot applications.

Which one is better YAML file or the Properties file and the different ways to load the YAML file in Spring boot.

Whether YAML or Properties file is better depends on your personal preference and the complexity of your configuration. Here are some considerations

1. Readability YAML provides a more human-readable and structured format, allowing for nested data structures and indentation. Properties files use a simple key-value pair structure.

2. Complex Configurations YAML is more suitable for complex configurations with nested properties or arrays. It supports multi-line values and hierarchical structures.

3. Simplicity Properties files are simpler and more familiar to developers who are accustomed to key-value configurations.

4. Spring Boot’s Default Spring Boot favors the use of YAML by default for external configuration, but it also supports Properties files.

To load a YAML file in Spring Boot, you can use the following methods

1. Using `@ConfigurationProperties` Annotate a class with `@ConfigurationProperties` and provide the YAML file’s path in the `value` attribute. Spring Boot will bind the YAML properties to the annotated class.

2. Using `@PropertySource` Use `@PropertySource` in conjunction with `@Configuration` to load a YAML file. Specify the YAML file’s location using the `value` attribute, and Spring Boot will load the properties into the environment.

3. Using `application.yml` or `application.yaml` By default, Spring Boot looks for the `application.yml` or `application.yaml` file in the classpath and automatically loads its properties into the application context.

These methods allow you to load YAML files and access their properties in a Spring Boot application, providing flexibility in configuring and customizing your application’s behavior.

Conclusion

As we step into 2023, it’s evident that a strong foundation in core Java is no longer adequate. A substantial portion of interview questions now revolves around Spring Boot and Microservices. By thoroughly exploring these questions, you not only enhance your knowledge but also prepare yourself for the essential topics in today’s tech landscape. Don’t forget to check out our related article on Spring Boot for further insights. Stay ahead in your career by keeping up with these evolving technologies.