100 Spring Boot interview questions and answers for 2025

Spring Boot is a refined Java application framework that provides a simplified and flexible method for constructing robust and scalable systems. As its popularity grows, so does the demand for skilled developers familiar with the framework. If you're a job seeker preparing for an interview or even a hiring manager exploring hiring software engineers, it's important to be well-versed in the framework.

We'll be exploring the top 100 Spring Boot interview questions that will be useful for novices and seasoned professionals looking to land a Spring Boot developer job. The same questions can be used by recruiters seeking Spring Boot experts.

Basic Spring Boot interview questions and answers

What is Spring Boot, and how does it differ from Spring Framework?

Spring Boot is a framework designed to simplify the development of Spring-based applications. It builds upon the Spring Framework, providing a convention-over-configuration approach and auto-configuration capabilities.

Unlike the Spring Framework, which requires explicit configuration, Spring Boot aims to minimize boilerplate code and provides defaults for various components. This makes it easier to get started with Spring-based applications.

Explain the benefits of using Spring Boot for application development.

Some benefits of using Spring Boot for application development include:

- Simplified setup and configuration through auto-configuration and starter dependencies.
- Reduced boilerplate code, enabling developers to focus more on application logic.
- Embedded server support, allowing applications to be run as standalone JAR files.
- Enhanced testability through the provision of t est utilities and annotations.

What are the key features of Spring Boot?

Key features of Spring Boot include:

Auto-configuration: Automatically configures Spring-based applications based on dependencies and defaults.

Starter dependencies: Pre-packaged dependencies that simplify the setup of specific application features or technologies.

Developer tools: Tools that enhance developer productivity such as automatic application restarts and live reload.

Actuator: Provides endpoints for monitoring and managing applications at runtime.

Explain the concept of Spring Boot starters and provide an example.

In the context of Spring Boot, Starters are a set of convenient dependency management providers that one can include in a Spring Boot application. Starters are a collection of dependency descriptors, which can help simplify your dependency management.

For instance, if you want to get started with Spring JPA, you just have to include the spring-boot-starter-data-jpa dependency and everything required for it (like Hibernate, Spring Data, etc.) will be added to your application.

Here's an example of what the Spring Boot Starter for JPA might look like in a pom.xml file:

```
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
</dependencies>
```

By including this dependency, Spring Boot provides all the required dependencies for creating a JPA application.

What is the purpose of the @SpringBootApplication annotation?

The @SpringBootApplication annotation is a convenience annotation provided by Spring Boot. It serves as the entry point for the Spring Boot application. It combines three commonly used annotations: @Configuration, @EnableAutoConfiguration, and @ComponentScan.

With @SpringBootApplication, developers can enable auto-configuration, component scanning, and configuration properties in a single step.

What is the default port number for a Spring Boot application?

The default port number for a Spring Boot application is 8080. However, you can change it by specifying the desired port number in the application's configuration file (e.g., application.properties or application.yml) using the property server.port.

How can you enable the auto-configuration feature in Spring Boot?

Auto-configuration is enabled by default in Spring Boot. It leverages the classpath and the defined dependencies to automatically configure the application. Spring Boot analyzes the dependencies and uses their presence to configure various components such as data sources, web servers, and messaging systems.

If needed, you can disable specific auto-configuration classes or customize the configuration by providing your own beans.

Explain the concept of starters in Spring Boot.

Starters in Spring Boot are a set of dependencies that make it easier to configure and use specific features or technologies in an application. They encapsulate the required dependencies and configurations, allowing developers to add them to their projects with minimal effort.

For example, the spring-boot-starter-web starter includes all the necessary dependencies for building web applications including the Spring MVC framework, embedded web server, and JSON support.

How does Spring Boot handle external configuration?

Spring Boot provides multiple ways to handle external configurations. It supports property files (application.properties or application.yml) that can be placed in various locations including the classpath, file system, or external directories.

Spring Boot also supports environment variables, command-line arguments, and the use of profiles for different deployment environments. The configuration values can be accessed using the @Value annotation or by binding them to Java objects using the @ConfigurationProperties annotation

What is the purpose of the application.properties (or application.yml) file?

The application.properties or application.yml file is used for external configuration in a Spring Boot application. It allows developers to specify various properties and their values to configure the application.

These properties can control various aspects of the application such as server port, database connection details, logging configuration, and much more. The properties file can be placed in the classpath or other predefined locations, and Spring Boot will automatically load and apply the configuration during application startup.

Describe the Spring Boot auto-configuration mechanism.

The Spring Boot auto-configuration mechanism automatically configures the Spring application based on the dependencies present in the classpath. It uses the concept of conditionals to determine which beans and configurations should be enabled or disabled.

By analyzing the classpath and the available configuration, Spring Boot can provide sensible defaults and reduce the need for explicit configuration. This makes it easier to start and configure a Spring application.

What is the purpose of the @Component annotation in Spring Boot?

The @Component annotation is a core annotation from the Spring Framework and is also used in Spring Boot. It is a generic stereotype annotation used to mark a class as a Spring-managed component.

Components are auto-detected by Spring and can be used for dependency injection and component scanning. The @Component annotation serves as a base annotation for more specific annotations like @Repository, @Service, and @Controller.

Explain the difference between @Component, @Repository, @Service, and @Controller annotations in Spring Boot.

@Component: It is a generic stereotype annotation used to mark a class as a Spring-managed component. It is a broad and generic term that can be used for any type of Spring-managed component.

@Repository: It is a specialized form of @Component used to indicate that a class is a repository or data access component. It typically encapsulates database operations and exception translation.

@Service: It is a specialized form of @Component used to indicate that a class is a service component. It encapsulates business logic and is often used as an intermediate layer between controllers and repositories.

@Controller: It is a specialized form of @Component used to indicate that a class is a web controller component. It handles incoming requests, performs business logic, and prepares the response to be sent back to the client.

What is the role of the @Autowired annotation in Spring Boot?

The @Autowired annotation is used for dependency injection in Spring Boot. When applied to a field, setter method, or constructor, it allows Spring to automatically resolve and inject the required dependencies.

By using @Autowired, developers don't need to manually instantiate and wire dependencies. Spring Boot scans the application context for beans matching the required type and injects them automatically.

How can you implement logging in a Spring Boot application?

In a Spring Boot application, logging is typically implemented using a logging framework such as Logback or Log4j2. Spring Boot provides a default logging configuration out of the box.

You can configure logging levels, appenders, and log formats using the application.properties or application.yml file. Additionally, you can include the desired logging framework dependencies in your project's build configuration and use the framework's APIs to perform logging within your application code.

What is the purpose of the SpringApplication.run() method?

The SpringApplication.run() method is used to bootstrap and launch a Spring Boot application. It is typically invoked from the main method of the application's entry point class.

The run() method initializes the Spring application context, performs auto-configuration, starts the embedded server, and starts the application lifecycle. It returns an instance of the ApplicationContext, allowing access to the application context and its beans.

What is Spring Boot CLI?

Spring Boot Command Line Interface (CLI) is a command line tool that you can use to run and test Spring Boot applications from a command prompt. It provides a fast way to get Spring applications up and running. The CLI incorporates spring scripts into the unix-based shell to launch the boot applications.

Some of the advantages of using Spring Boot CLI are:

- It allows you to write your application using Groovy, which is a more succinct and expressive alternative to Java.
- It automatically includes useful external libraries whenever possible. For example, if you're writing a web application and importing classes such as @RestController, the CLI will automatically provide a dependency for Spring MVC.
- You can use various commands for different operations like run (to run the application), test (to test the application), jar (to create a jar file), init (to create a basic Java or Groovy project), etc.

How does Spring Boot handle data validation?

In Spring Boot, data validation can be performed using various mechanisms. One common approach is to use the validation annotations provided by the Bean Validation API, such as @NotNull, @Size, and @Pattern, on the fields of model objects.

By including the necessary validation annotations, Spring Boot automatically validates the input data and generates validation errors. These errors can be handled using BindingResult or Errors objects. Additionally, custom validation logic can be implemented by creating custom validation classes and methods.

What is the purpose of the @RequestMapping annotation in Spring Boot?

The @RequestMapping annotation is used to map HTTP requests to specific handler methods in a Spring Boot application. It is applied at the method or class level to define the URL patterns that should trigger the execution of the annotated method.

@RequestMapping allows developers to specify various attributes, such as the HTTP method (GET, POST, etc.), request parameters, headers, and more to further refine the mapping.

How does Spring Boot integrate with containerization platforms like Docker and Kubernetes?

Spring Boot integrates seamlessly with containerization platforms like Docker and Kubernetes. You can package a Spring Boot application as a Docker image by creating a Dockerfile that includes the necessary dependencies and configurations.

The image can be built and deployed to a containerization platform like Docker Swarm or Kubernetes. Spring Boot also provides features like externalized configuration and health indicators which can be leveraged by container orchestration platforms for efficient management and scaling of the application.

Explain the concept of message-driven microservices using Spring Boot and Apache Pulsar.

Message-driven microservices using Spring Boot and Apache Pulsar leverage the publish-subscribe messaging pattern to enable loosely coupled and scalable communication between microservices. Apache Pulsar acts as the messaging system, and Spring Boot provides the necessary abstractions for consuming and producing messages.

With Pulsar's messaging features and Spring Boot's integration, you can implement event-driven architectures where microservices communicate asynchronously through messages. This ensures decoupling and fault tolerance.

What is the purpose of the @Value annotation in Spring Boot?

The @Value annotation is used to inject values from properties files, environment variables, or other sources into Spring-managed beans. It can be applied to fields, methods, or constructor parameters.

With @Value, developers can easily access and use configuration properties or other values within their application code. The values can be specified directly or referenced using SpEL (Spring Expression Language) expressions.

Describe the role of the CommandLineRunner and ApplicationRunner interfaces in Spring Boot.

In Spring Boot, the CommandLineRunner and ApplicationRunner interfaces are used for performing specific tasks during the application startup process. When implemented, these interfaces provide a callback method (run()) that gets executed once the application context is initialized.

They are particularly useful for performing tasks like data initialization, cache population, or other one-time setup operations. The main difference between them is that CommandLineRunner receives the application's command-line arguments as a parameter, while ApplicationRunner receives an ApplicationArguments object.

How can you implement pagination in a Spring Boot application?

To implement pagination in a Spring Boot application, you can utilize features provided by libraries like Spring Data JPA or Spring Data MongoDB. They offer built-in support for pagination through the use of Pageable objects and repository methods.

You can retrieve a subset of data from a larger dataset by specifying the page number, page size, and sort criteria. The result is typically returned as a Page object that contains the requested data along with metadata such as total elements, total pages, and more.

Explain the concept of bean scopes in Spring Boot.

Bean scopes define the lifecycle and visibility of Spring-managed beans in a Spring Boot application. The following are the commonly used bean scopes:

Singleton (default): Only one instance of the bean is created and shared across the entire application context.

Prototype: A new instance of the bean is created each time it is requested.

Request: A new instance of the bean is created for each HTTP request. It is only applicable in a web application context.

Session: A new instance of the bean is created for each user session. It is only applicable in a web application context.

Custom scopes: Spring Boot allows defining custom bean scopes by implementing the Scope interface and registering them in the application context.

What is the purpose of the @Qualifier annotation in Spring Boot?

The @Qualifier annotation in Spring is used to disambiguate bean references when we have multiple beans of the same type defined in the Spring container. It is used in scenarios where a given type has more than one implementation and we need to inject a specific implementation.

By default, Spring uses the by-type autowiring mechanism. This means that if we have more than one bean of the same type, Spring will throw a NoUniqueBeanDefinitionException because it won't know which one to autowire.

The @Qualifier annotation can be used in conjunction with @Autowired to specify which exact bean should be wired, by providing the name of the bean as the qualifier value.

How does Spring Boot handle exception logging and error handling?

In Spring Boot, exception logging and error handling can be configured using various mechanisms. Spring Boot automatically provides a default error page that displays a standardized error message for unhandled exceptions.

However, you can customize the error-handling behavior by implementing exception handlers using the @ControllerAdvice annotation and handling specific exceptions in dedicated methods.

Additionally, you can configure logging frameworks to capture and log exceptions with desired levels of detail and appenders.

Describe the purpose and usage of the @RestControllerAdvice annotation.

The @RestControllerAdvice annotation is a specialized form of the @ControllerAdvice annotation in Spring Boot. It combines the functionality of @ControllerAdvice and @ResponseBody, making it convenient for implementing global exception handling in RESTful APIs.

By using @RestControllerAdvice, you can define exception handlers that handle exceptions thrown by any @RequestMapping or @RestController method within the application. The exception handlers can return error responses in JSON or other supported formats.

What is the purpose of the @ConfigurationProperties annotation in Spring Boot?

The @ConfigurationProperties annotation is used to bind external configuration properties to Spring-managed beans. By annotating a bean class with @ConfigurationProperties and specifying a prefix, you can map properties with matching names to the fields or setter methods of the bean.

Spring Boot will automatically bind the values from the configuration sources to the corresponding bean properties. The annotation simplifies the retrieval and usage of configuration properties within your application.

Describe the purpose and usage of the @DynamicPropertySource annotation in Spring Boot testing.

The @DynamicPropertySource annotation in Spring Boot testing allows you to dynamically define and modify configuration properties during the test execution. You can use this annotation in conjunction with the TestPropertyValues class to set or override properties based on dynamic values or test conditions.

This provides flexibility in configuring the environment for testing and allows you to simulate different scenarios or configurations during testing.

What is the purpose of the @TransactionalEventListener annotation in Spring Boot?

The @TransactionalEventListener annotation in Spring Boot lets you listen to transactional events and perform actions based on those events. You can use this annotation on methods that should be invoked when a specific transactional event occurs such as before or after a transaction is committed or rolled back.

The @TransactionalEventListener annotation provides a convenient way to handle domain-specific logic or side effects based on transactional events in a Spring Boot application.

What is the purpose of the @Scheduled annotation in Spring Boot?

The @Scheduled annotation is used to configure scheduled tasks in a Spring Boot application. Applying this annotation to a method enables you to specify the schedule at which the method should be executed.

The schedule can be defined using various options such as fixed-rate, fixed-delay, or cron expressions. Spring Boot automatically detects and executes the scheduled methods based on the specified schedule.

Describe the role of the @Profile annotation in Spring Boot.

The @Profile annotation is used to activate or deactivate specific configuration components or beans based on the current environment or profile in a Spring Boot application. Annotating a class or method with @Profile and specifying the desired profile name lets you control when that component or bean should be active. This allows you to have different configurations for different deployment environments such as development, testing, or production.

What is the purpose of Spring Boot's dynamic reloading and how does it work?

Spring Boot's dynamic reloading feature allows you to make changes to the application code or resources without the need to restart the entire application. It improves development productivity by automatically reloading the modified classes or resources on the fly.

The dynamic reloading feature uses class reloading mechanisms provided by the underlying JVM, such as Java Instrumentation API or custom class loaders, to reload the changed classes while preserving the application's state.

Explain the concept of externalized logging in Spring Boot using Logback or Log4j2.

Externalized logging in Spring Boot allows you to configure and customize logging behavior without modifying the application code. Logback or Log4j2 can be used as the underlying logging framework.

The configuration is typically done in an external configuration file, such as logback.xml or log4j2.xml, which can be placed in the classpath or specified using the logging.config property. The externalized logging configuration file provides flexibility in defining log levels, appenders, formatters, and other logging-related properties.

What is the purpose of the @ModelAttribute annotation in Spring Boot?

The @ModelAttribute annotation is used in Spring Boot to bind request parameters or form data to method parameters or model attributes. It can be applied to method parameters or method return values.

When applied to method parameters, the @ModelAttribute annotation binds the incoming request parameters or form data to the corresponding method parameters. When applied to method return values, it binds the method's return value to a model attribute, making it available in the view for rendering.

Explain the concept of reactive messaging with Spring Boot and Apache Kafka Streams.

Reactive messaging with Spring Boot and Apache Kafka Streams enables the building of real-time streaming applications that react to events and reactively process data streams. Spring Cloud Stream provides abstractions to integrate Spring Boot applications with Kafka Streams.

With @StreamListener annotations, you can consume Kafka topics as reactive streams and perform processing operations using the reactive programming model. This approach facilitates the development of scalable and resilient streaming applications.

Describe the purpose and usage of the @Transactional(propagation = Propagation.NESTED) annotation.

The @Transactional(propagation = Propagation.NESTED) annotation is used to define a nested transactional scope in a Spring Boot application. When a method is annotated with this annotation, a nested transaction is created within the current transaction.

The nested transaction behaves as an independent transaction and can be rolled back separately from the outer transaction. If the nested transaction fails, only the changes made within the nested transaction are rolled back, while the outer transaction remains unaffected.

What is the purpose of the @DataJpaTest annotation in Spring Boot testing?

The @DataJpaTest annotation is used to configure and customize the testing environment for JPA repositories in a Spring Boot application. When applied to a test class, it sets up an in-memory database, configures Spring Data JPA, and loads only the necessary components for testing JPA repositories.

@DataJpaTest provides a lightweight and isolated environment for testing JPA-related functionality without requiring a full application context or a real database connection.

Describe the purpose and usage of the Spring Boot Admin Server for monitoring and managing applications.

The Spring Boot Admin Server is a tool that provides a web-based interface for monitoring and managing multiple Spring Boot applications in a centralized manner. It collects and displays various metrics, health statuses, and other information about the registered Spring Boot applications.

The Admin Server allows you to view and manage application details, monitor JVM metrics, and receive alerts on specific conditions. It simplifies the monitoring and management of Spring Boot applications in a production environment.

Intermediate Spring Boot interview questions and answers

Differentiate between Spring MVC and Spring Boot.

Spring MVC is a framework for building web applications using the Model-View-Controller (MVC) architectural pattern. It provides features for handling requests, managing controllers, rendering views, and managing data flow.

Spring Boot, on the other hand, is an opinionated framework built on top of Spring that aims to simplify the setup and configuration of Spring applications. It provides out-of-the-box defaults and auto-configuration, reducing the need for manual configuration and boilerplate code.

What is the role of the @RestController annotation in Spring Boot?

The @RestController annotation is used to define a RESTful controller in a Spring Boot application. It combines the functionality of the @Controller and @ResponseBody annotations, simplifying the process of building RESTful APIs by automatically serializing the return values of methods into JSON or XML responses.

How can you implement exception handling in a Spring Boot application?

Exception handling in Spring Boot can be implemented using the @ControllerAdvice annotation. By creating a class annotated with @ControllerAdvice and defining methods annotated with @ExceptionHandler, you can handle specific exceptions and return appropriate responses.

You can also use the @ResponseStatus annotation to specify the HTTP status code for the response.

Explain the concept of dependency injection in Spring Boot.

Dependency injection is a core concept in Spring Boot. It allows objects to be loosely coupled by providing their dependencies from external sources. Spring Boot uses inversion of control (IoC) and the dependency injection pattern to manage dependencies.

Spring Boot automatically resolves and injects the required dependencies at runtime by annotating classes with appropriate annotations such as @Autowired.

How does Spring Boot support database operations?

Spring Boot provides excellent support for database operations through its integration with Spring Data JPA. By defining entities and repositories, you can perform CRUD (Create, Read, Update, Delete) operations on databases with minimal boilerplate code.

Spring Boot automatically configures the database connection and transaction management, and provides powerful querying capabilities.

Describe the role of the Spring Boot Actuator.

Spring Boot Actuator is a feature that provides insight into the runtime of a Spring Boot application. It offers a set of production-ready endpoints that expose information about application health, metrics, environment, logging, and more.

The Actuator enables monitoring and management of the application, making it easier to understand and troubleshoot in production environments.

How can you implement caching in a Spring Boot application?

Caching in a Spring Boot application can be implemented using the @Cacheable, @CacheEvict, and other cache-related annotations provided by the Spring Framework. Adding these annotations to methods lets you cache the results and improve performance. Spring Boot integrates with popular caching providers like Ehcache, Hazelcast, and Redis.

What is the purpose of the @Scheduled annotation in Spring Boot?

The @Scheduled annotation is used to schedule the execution of a method at fixed intervals or specific times. It allows you to define cron expressions or fixed delay/initial delay values. Spring Boot automatically triggers the annotated method based on the specified schedule, making it suitable for performing recurring tasks such as data synchronization or sending periodic notifications.

How can you enable cross-origin resource sharing (CORS) in a Spring Boot application?

To enable CORS in a Spring Boot application, you can use the @CrossOrigin annotation at the controller level or globally configure CORS using a WebMvcConfigurer bean. The annotation allows you to specify the allowed origins, HTTP methods, headers, and other CORS-related settings. Enabling CORS ensures that web browsers can make requests to your application from different domains.

Explain the concept of profiles in Spring Boot.

Profiles in Spring Boot allow you to define different configurations for different environments or scenarios. By using the @Profile annotation on classes or methods, you can specify which profiles should be active for the corresponding beans or configurations.

Profiles enable you to have different property values, component configurations, or dependencies based on the active profile. This facilitates easy deployment and testing across different environments.

Explain the concept of the Spring Boot Actuator and its major features.

Spring Boot Actuator provides a set of production-ready features and endpoints that help monitor and manage a Spring Boot application.

Its major features include health checks, which provide information about the application's health; metrics, which gather and expose various runtime metrics; info, which displays custom application information; logging, which allows changing log levels at runtime, and many more endpoints for managing and understanding the application in a production environment.

How can you integrate Spring Security in a Spring Boot application?

To integrate Spring Security in a Spring Boot application, you need to add the appropriate dependencies and configure security settings. Spring Security provides comprehensive authentication and authorization mechanisms.

You can configure security rules using Java configuration or annotations, define user roles and permissions, and customize authentication providers, such as in-memory authentication, database-backed authentication, or integration with external identity providers.

Describe the role of the @Transactional annotation in Spring Boot.

The @Transactional annotation is used to mark a method or class for transaction management in Spring Boot. It ensures that the annotated method or all methods within the annotated class are executed within a transactional context.

The @Transactional annotation manages the transaction boundaries, rollback behavior, and other transactional aspects to ensure data consistency and integrity.

How does Spring Boot handle internationalization (i18n) and localization (l10n)?

Spring Boot automatically resolves the appropriate message based on the user's locale, making it convenient to build multi-language applications. It provides support for internationalization and localization through properties files and the use of the MessageSource interface.

By defining message bundles for different locales and configuring the message source, you can easily retrieve and display localized messages in your application.

What is the purpose of the @RestControllerAdvice annotation?

The @RestControllerAdvice annotation combines the functionalities of @ControllerAdvice and @ResponseBody annotations. It is used to define a global exception handler for RESTful controllers in a Spring Boot application.

Annotated classes can contain exception-handling methods annotated with @ExceptionHandler which handles exceptions thrown within any @RestController in the application. These methods can return custom error responses or perform other actions based on the exception type.

Explain the concept of Spring Data REST and its advantages.

Spring Data REST is a project built on top of Spring Data, it takes the features of Spring HATEOAS and Spring Data to build Spring MVC-based RESTful services with less code. With Spring Data REST, you can leverage your Spring Data repositories and convert them into full-featured RESTful services with ease.

Some of its advantages are:

Rapid Development: With Spring Data REST, a great deal of your HTTP resource implementation time can be saved. It's quick and easy to build a RESTful service with full CRUD functionality.

Data Access: It leverages Spring Data's repositories and provides seamless, RESTful access to your data model.

HAL Browser: Spring Data REST includes support for the HAL Browser, allowing users to navigate, create, update, and delete resources directly from their web browsers.

Search Support: It has built-in support for searches. Custom repository methods are automatically exposed as HTTP resources.

How can you implement file upload and download functionality in a Spring Boot application?

File upload and download functionality can be implemented in a Spring Boot application by configuring multipart file handling. By using the MultipartFile object as a method parameter, Spring Boot automatically binds uploaded files to it.

For file download, you can return the file as a response with appropriate headers. Additionally, you can leverage storage services like Amazon S3 or Azure Blob Storage for file storage and retrieval.

Describe the purpose and usage of the @Async annotation in Spring Boot.

The @Async annotation is used to indicate that a method should be executed asynchronously. When a method is annotated with @Async, Spring Boot runs it in a separate thread from a task executor, allowing the caller to continue execution without waiting for the asynchronous method to complete.

This annotation is useful for offloading time-consuming tasks, improving performance, and providing a more responsive user experience.

What is the role of the embedded servlet container in Spring Boot?

The embedded servlet container in Spring Boot allows you to run web applications without the need for a separate web server. It provides a lightweight servlet container, such as Tomcat, Jetty, or Undertow, that is embedded within the application.

Spring Boot automatically configures and starts the embedded servlet container, simplifying the deployment and execution of web applications.

How can you implement request and response logging in a Spring Boot application?

Request and response logging in a Spring Boot application can be implemented using filters or interceptors. Creating a custom filter or interceptor lets you intercept incoming requests and outgoing responses and log their details, such as headers, payloads, and other relevant information.

Spring Boot allows you to register these filters or interceptors in the application's configuration, enabling centralized logging across the application.

Explain the concept of reactive data access in Spring Boot using Spring Data R2DBC.

Reactive data access in Spring Boot allows you to build non-blocking and efficient applications that handle a large number of concurrent requests. Spring Data R2DBC provides reactive database access by integrating with R2DBC (Reactive Relational Database Connectivity).

It enables you to perform asynchronous database operations using reactive programming paradigms, such as Flux and Mono, providing better scalability and responsiveness compared to traditional blocking database access.

How can you perform asynchronous messaging using Spring Boot and RabbitMQ?

Asynchronous messaging using Spring Boot and RabbitMQ can be achieved by integrating the Spring AMQP (Advanced Message Queuing Protocol) library. By configuring the RabbitMQ connection details and using the appropriate annotations and components, you can send and receive messages asynchronously.

Spring Boot provides abstractions like @RabbitListener for message consumption and RabbitTemplate for message production. They make it easy to implement asynchronous messaging patterns like publish-subscribe and request-reply.

Describe the purpose and usage of the @Conditional annotation in Spring Boot.

The @Conditional annotation in Spring Boot allows you to conditionally activate or deactivate beans or configurations based on specific conditions. By annotating a bean or configuration class with @Conditional and providing a condition class implementing the

Condition interface, you can control whether the bean or configuration should be created and registered based on runtime conditions. This enables flexible configuration based on environment, properties, or other factors.

What is the purpose of the @SpringBootTest annotation in Spring Boot testing?

The @SpringBootTest annotation is used to bootstrap a Spring Boot application context for testing purposes. It allows you to load the entire application context, including all configurations and beans, during integration tests.

@SpringBootTest provides features like auto-configuration, dependency injection, and easy access to application-specific components, enabling comprehensive testing of Spring Boot applications.

Explain the concept of Spring Boot's actuator health checks and customizing health indicators.

Spring Boot's actuator health checks are endpoints provided by the Actuator that give insights into the application's health. By default, health indicators check the overall system health. You can customize them by implementing the HealthIndicator interface and registering them with the application context.

Custom health indicators allow you to monitor specific aspects of the application's health such as database connectivity, external service availability, or custom health checks.

How can you secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT)?

You can secure REST APIs in a Spring Boot application using JSON Web Tokens (JWT) by integrating Spring Security and JWT libraries. Spring Security provides mechanisms for authentication and authorization, while JWT facilitates token-based authentication.

By configuring Spring Security filters, implementing authentication and authorization providers, and validating JWT tokens, you can protect your REST APIs and control access based on user roles and permissions.

Describe the purpose and usage of the @EntityScan annotation in Spring Boot.

The @EntityScan annotation is used to specify the base packages to scan for entity classes in a Spring Boot application. When using JPA (Java Persistence API) with Spring Boot, @EntityScan helps the JPA provider locate and manage entity classes.

By default, Spring Boot scans the package of the application's main class and its sub-packages. However, if entity classes are located in different packages, you need to use @EntityScan to include those packages.

What is the purpose of the @Retryable annotation in Spring Boot?

The @Retryable annotation is used to specify that a method should be retried if it fails due to specified exceptions. Adding @Retryable and configuring the desired retry behavior enables Spring Boot to automatically retry the method when exceptions occur.

This can be useful for handling transient errors, such as network timeouts or temporary resource unavailability, and ensuring the successful execution of critical operations.

Explain the concept of auto-reconfiguration in Spring Boot and its limitations.

Auto-reconfiguration in Spring Boot is a feature that automatically configures certain components and dependencies based on the classpath and available resources. It simplifies the configuration process by detecting and configuring components like data sources, messaging brokers, and caches.

Auto-reconfiguration has limitations when it comes to complex or custom configurations. It may not always provide the desired configuration out of the box. In such cases, manual configuration may be required.

How can you implement distributed caching in a Spring Boot application using Hazelcast or Redis?

To implement distributed caching in a Spring Boot application with Hazelcast or Redis, you can leverage the respective cache providers' integration libraries. Configuring the cache manager and cache-related settings lets you enable distributed caching.

Spring Boot simplifies the setup and configuration process by providing auto-configuration support for both Hazelcast and Redis. Additionally, you can annotate methods with cache-related annotations like @Cacheable or @CacheEvict to cache and retrieve data efficiently.

Advanced Spring Boot interview questions

How can you create a Spring Boot application using Gradle?

To create a Spring Boot application using Gradle, follow these steps:

- Set up a new Gradle project or add Spring Boot dependencies to an existing Gradle project.
- Make sure you have the required plugins configured in the build.gradle file, such as the org.springframework.boot and io.spring.dependency-management plugins.
- Define the necessary dependencies in the dependencies section of the build.gradle file. Specify the desired Spring Boot starter dependencies.
- Create the main application class and annotate it with @SpringBootApplication.
- Implement the application logic within the main application class or other components.
- Use the Gradle command line or an IDE plugin to build and run the application.

How can you customize the default error pages in a Spring Boot application?

To customize the default error pages in a Spring Boot application, you can create an error page template or controller method that handles the error. Defining an error template with the appropriate name and placing it in the src/main/resources/templates/error directory enables Spring Boot to automatically render that template for the corresponding error status code.

Alternatively, you can create a controller method with @ExceptionHandler annotation to handle specific exceptions and return a custom error response.

How can you create a Spring Boot application using Maven?

To create a Spring Boot application using Maven, follow these steps:

- Set up a new Maven project or add Spring Boot dependencies to an existing Maven project.
- Ensure that the project's dependencies include the spring-boot-starter-parent as the parent project.
- Define the necessary dependencies in the project's pom.xml file, such as spring-boot-starter-web for web applications.
- Create the main application class and annotate it with @SpringBootApplication.
- Implement the application logic within the main application class or other components.
- Use the Maven command line or an IDE plugin to build and run the application.

How can you implement security in a Spring Boot application?

Security can be implemented in a Spring Boot application by adding the appropriate dependencies, such as spring-boot-starter-security, and configuring the security settings. This can be done by creating a security configuration class that extends WebSecurityConfigurerAdapter and overriding its methods to define authentication and authorization rules.

Additionally, you can customize the login page, handle logout, and secure specific endpoints using annotations like @EnableWebSecurity and @EnableGlobalMethodSecurity.

What are the different deployment options for a Spring Boot application?

Spring Boot applications can be deployed in various ways including:

Standalone JAR: Packaging the application as a self-contained executable JAR file with an embedded servlet container like Tomcat or Jetty.

WAR deployment: Packaging the application as a traditional WAR file and deploying it to a servlet container.

Docker: Containerizing the application using Docker and running it on Docker containers.

Cloud platforms: Deploy the application to cloud platforms like AWS, Azure, or Google Cloud using platform-specific deployment options such as AWS Elastic Beanstalk or Azure App Service.

Describe the process of creating a RESTful API using Spring Boot.

To create a RESTful API using Spring Boot, you can follow these steps:

- Define your domain model and business logic.
- Create a Spring MVC controller class and define handler methods annotated with @RequestMapping or other mapping annotations.
- Implement the required CRUD operations within the handler methods using appropriate annotations like @GetMapping, @PostMapping, etc.
- Customize the request and response handling with annotations such as @RequestBody to map request payloads and @ResponseBody to define the response body.
- Configure additional features like exception handling, input validation, and security, if required.
- Run the Spring Boot application and the API endpoints will be accessible based on the mappings defined in the controller.

How does Spring Boot integrate with messaging systems such as RabbitMQ?

Spring Boot provides integration with messaging systems like RabbitMQ through the Spring AMQP project. To integrate RabbitMQ with a Spring Boot application, you can include the spring-boot-starter-amqp dependency and configure the necessary properties in the application's configuration file (application.properties or application.yml).

You can use the RabbitTemplate class to send messages to RabbitMQ and consume messages using @RabbitListener annotations on appropriate methods.

Explain the concept of Spring Boot Data JPA and provide an example.

Spring Boot Data JPA is a sub-project of Spring Data that provides enhanced support for JPA (Java Persistence API)-based repositories in Spring Boot applications. It simplifies the implementation of the data access layer by automatically generating the boilerplate code for common database operations.

For example, by defining a JPA entity class and extending JpaRepository, you can get CRUD operations for that entity without writing any additional code. You can also define custom queries using method name conventions or @Query annotations

How can you handle large file uploads in a Spring Boot application?

To handle large file uploads in a Spring Boot application, you can configure the maximum file size limit in the application's properties file by setting the spring.servlet.multipart.max-file-size and spring.servlet.multipart.max-request-size properties to appropriate values.

Additionally, you can use the MultipartFile parameter in the controller method to receive the uploaded file and process it as needed.

What is the purpose of Spring Boot Actuator endpoints?

Spring Boot Actuator endpoints provide insights into the internals of a Spring Boot application such as health status, metrics, environment information, and more. These endpoints expose management and monitoring capabilities over HTTP or other protocols, allowing you to monitor and manage the application in production. Actuator endpoints can be customized and secured based on the specific requirements of the application.

Explain the concept of Spring Boot Actuator metrics and monitoring.

Spring Boot Actuator metrics allow you to collect and monitor various application metrics such as HTTP request counts, response times, JVM memory usage, and database connection pool metrics.

Actuator metrics are collected by integrating with metrics libraries like Micrometer and can be exposed through various endpoints such as /actuator/metrics. These metrics can be visualized using monitoring tools like Prometheus, Grafana, or the built-in Actuator endpoints.

How can you implement microservices architecture using Spring Boot?

To implement a microservices architecture using Spring Boot, you can follow these steps:

- Identify the different business capabilities and boundaries of your application.
- Design and develop each microservice as a separate Spring Boot application, encapsulating a specific business capability.
- Use lightweight communication mechanisms like REST or messaging for inter-service communication.
- Implement service discovery and registration using tools like Netflix Eureka or HashiCorp Consul.
- Apply fault tolerance and resilience patterns like circuit breakers (Hystrix) and distributed tracing (Sleuth) for better reliability.
- Deploy and manage microservices using containerization platforms like Docker and orchestration tools like Kubernetes.

Describe the role of the Spring Cloud Netflix stack in a Spring Boot application.

The Spring Cloud Netflix stack provides integration with various Netflix OSS components to simplify the development of distributed systems in a Spring Boot application. It includes modules like Eureka for service discovery, Ribbon for client-side load balancing, Hystrix for fault tolerance, and Zuul for API gateway functionality.
These components enable developers to build scalable and resilient microservices architectures by providing out-of-the-box solutions for common distributed system challenges.

What is the purpose of Spring Boot DevTools and how does it enhance development productivity?

Spring Boot DevTools is a set of developer tools that enhance the development experience for Spring Boot applications. It provides features like automatic application restart on code changes, live reloading of static resources, and enhanced logging during development.

DevTools helps in reducing the development turnaround time by eliminating the need for manual restarts and providing quick feedback on code changes.

How can you implement distributed tracing in a Spring Boot application using Spring Cloud Sleuth?

To implement distributed tracing in a Spring Boot application with Spring Cloud Sleuth, you can include the necessary dependencies like spring-cloud-starter-sleuth. Sleuth integrates with popular distributed tracing systems like Zipkin or Jaeger.

Once configured, Sleuth automatically adds trace and span identifiers to the application's logs and propagates them across different services. This allows you to trace the flow of requests across multiple services and analyze performance bottlenecks.

Explain the concept of reactive programming in Spring Boot with Spring WebFlux.

Reactive programming in Spring Boot with Spring WebFlux is based on the Reactive Streams specification and enables non-blocking, event-driven programming for building scalable and resilient applications.

Spring WebFlux provides an alternative to the traditional Servlet-based programming model and allows developers to handle requests asynchronously using reactive types like Mono and Flux. This approach is well-suited for handling high concurrency and building reactive systems that can handle a large number of concurrent connections with limited resources.

How does Spring Boot integrate with Apache Kafka for event-driven architectures?

Spring Boot provides integration with Apache Kafka through the Spring Kafka project. You can include the spring-boot-starter-kafka dependency to get started.

Spring Kafka provides abstractions to produce and consume messages from Kafka topics using the KafkaTemplate and @KafkaListener annotations, respectively. Additionally, Spring Kafka integrates with Spring Boot's auto-configuration to simplify the configuration of Kafka-related properties.

Describe the purpose and usage of Spring Boot's caching abstraction.

Spring Boot's caching abstraction provides a convenient way to cache the results of expensive operations, reducing the response time and improving application performance. By using annotations like @Cacheable, @CachePut, and @CacheEvict, you can easily cache method results based on specified cache names or keys.

The caching abstraction supports various cache providers, such as Redis or Ehcache, and can be easily configured using the application's properties file.

How can you perform database migrations in a Spring Boot application using Flyway or Liquibase?

Include corresponding dependencies (flyway-core or liquibase-core) to perform database migrations in a Spring Boot application using Flyway or Liquibase. By placing the migration scripts in the classpath (src/main/resources/db/migration), Flyway or Liquibase will automatically execute the scripts during application startup.

These migration scripts allow you to manage database schema changes, versioning, and data initialization in a controlled manner.

Explain the concept of externalized configuration in Spring Boot and its benefits.

Externalized configuration in Spring Boot allows you to configure the application using external properties files, environment variables, or command-line arguments. This approach decouples the application configuration from the code, making it more flexible and easier to manage.

The externalized configuration enables the application to be deployed in different environments without modifying the code. Spring Boot provides a standardized and flexible way to read and use these external configurations.

How can you implement distributed session management in a Spring Boot application using Spring Session?

To implement distributed session management in a Spring Boot application using Spring Session, follow these steps:

- Include the necessary dependencies for Spring Session and a session store implementation like Redis or Hazelcast.
- Configure the session store details, such as the connection properties, in the application's configuration file (e.g., application.properties or application.yml).
- Enable Spring Session support by annotating your configuration class with @EnableRedisHttpSession (for Redis) or @EnableHazelcastHttpSession (for Hazelcast).
- Spring Session will automatically handle session creation, serialization, and synchronization with the session store, allowing session data to be shared across multiple instances of your application.

How can you implement serverless functions using Spring Boot and AWS Lambda?

You can use the spring-cloud-function-adapter-aws dependency to implement serverless functions using Spring Boot and AWS Lambda. You can deploy Spring Boot applications as serverless functions on AWS Lambda by creating a function bean and configuring the AWS Lambda handler.

The adapter takes care of the integration between Spring Cloud Function and AWS Lambda, allowing you to develop serverless functions using the familiar Spring Boot programming model.

How can you implement method-level security in a Spring Boot application?

To implement method-level security in a Spring Boot application, you can use the @PreAuthorize or @PostAuthorize annotations provided by Spring Security. Here's an example:

- Configure Spring Security in your application by including the necessary dependencies and configuration.
- Annotate the methods that require security checks with @PreAuthorize or @PostAuthorize.
- Specify the desired security expressions in the annotations to define the required conditions for method invocation.
- Spring Security will evaluate the expressions and allow or deny access to the methods based on the configured security rules.

How can you implement server-sent events (SSE) in a Spring Boot application?

To implement server-sent events (SSE) in a Spring Boot application, you can use the SseEmitter class provided by Spring Framework. Here's an example of how to implement SSE:

- Create a controller method that returns an SseEmitter object.
- In this method, use the SseEmitter to send events to the client.
- Use the send() method of SseEmitter to send events periodically or based on specific triggers.
- Set appropriate headers, such as Content-Type and Cache-Control, for SSE support.
- Register the SseEmitter as a handler method in your controller.

How can you integrate Spring Boot with OAuth 2.0 for secure authentication and authorization?

You can use the Spring Security OAuth2 module to integrate Spring Boot with OAuth 2.0. By configuring the appropriate OAuth 2.0 provider details and defining the client registration properties, Spring Boot can handle the authentication and authorization flow.

You can secure your endpoints by applying Spring Security annotations like @PreAuthorize or by using declarative configuration. This integration enables secure authentication and authorization using OAuth 2.0 standards.

How can you implement data caching in a Spring Boot application using the Spring Cache Abstraction?

To implement data caching in a Spring Boot application, you can leverage the Spring Cache Abstraction. Follow these steps:

- Enable caching support by annotating your configuration class with @EnableCaching.
- Add the desired cache implementation library, such as Ehcache or Caffeine, as a dependency.
- Annotate the methods that should be cached with @Cacheable and specify the cache name or key.
- Configure the cache properties, such as eviction policies and time-to-live, in the cache implementation's configuration file or using Spring Boot's properties.

How can you configure a custom error page in a Spring Boot application?

To configure a custom error page in a Spring Boot application, you can create a custom error controller and map it to a specific URL or error status code. Here's an example:

- Create a class implementing the ErrorController interface.
- Annotate the class with @Controller and, optionally, with @RequestMapping to specify the mapping URL or error status code.
- Implement a method that handles the error and returns the desired error page view or response.
- Register the custom error controller as a bean in the application context.
- Optionally, configure the error page mapping in the application.properties or application.yml file using properties like server.error.path or server.error.whitelabel.enabled.

How can you implement distributed tracing in a Spring Boot application using OpenTelemetry?

To implement distributed tracing in a Spring Boot application using OpenTelemetry, you can include the necessary OpenTelemetry dependencies such as open telemetry-API and an OpenTelemetry exporter. Configuring the exporter lets you send trace data to a distributed tracing system like Jaeger or Zipkin.

OpenTelemetry automatically instructs the application to capture and propagate trace information across different services, allowing you to trace the execution path of requests in a distributed system.

How can you implement asynchronous processing in a Spring Boot application?

Spring Boot provides support for asynchronous processing through the use of the @Async annotation and the TaskExecutor interface. To implement asynchronous processing, follow these steps:

- Configure a TaskExecutor bean in your application's configuration. This bean defines the thread pool or executor service used for executing asynchronous tasks.
- Annotate the methods that need to be executed asynchronously with the @Async annotation.
- Invoke the annotated methods from other parts of your application. The invocation will return a Future object which can be used to obtain the result of the asynchronous task or monitor its progress.

How can you enable HTTPS in a Spring Boot application?

To enable HTTPS in a Spring Boot application, you need to configure the appropriate SSL certificate and modify the application's configuration. Here are the general steps:

- Obtain an SSL certificate and private key.
- Configure the certificate and private key in the application's configuration, such as application.properties or application.yml, using properties like server.ssl.key-store and server.ssl.key-password.
- Set the server.ssl.enabled property to true to enable HTTPS.
- Optionally, configure other SSL-related properties like the SSL protocol and cipher suites.
- Restart the application for the changes to take effect.

Wrapping up

There is no doubt that the impact and importance of Spring Boot in the world of Java application development. As a developer or hiring manager, understanding the ins and outs of Spring Boot has become nothing short of essential. This extensive list of the top 100 Spring Boot interview questions and answers serves as a critical learning tool for aspiring boot engineers and a valuable resource for hiring managers in search of top-tier talent.

If you are a developer seeking a Spring Boot opportunity, let Turing be your guide. Turing connects talented developers like you with top US and Silicon Valley companies in need of expertise. We make the process of landing your dream job a breeze.

For hiring managers looking to infuse their teams with Spring Boot practitioners, Turing can offer a wealth of talented developers ready to hit the ground running. With a talent base that expands beyond Spring Boot to include other technologies like Kubernetes, Turing is ready to supply the exact technical skills you need to succeed on your projects.