

Spring Boot Concepts Overview



1. what is swagger and how to use it in spring boot
2. what is rest template and how to send response using it give code in simple
3. connect and response time in spring boot microservices
4. local transaction in microservices
5. failures in microservices and how to handle
6. Service mesh what is it?
7. redis integration with spring boot code in simple and use of redis
8. SB caching explain in simple and code in simple
9. how to deploy apps on Pivotal cloud foundry
10. Jenkins working and how to create pipeline
11. Docker working and how to create pipeline
12. some details about splunk and splunk query why is it needed.
13. Enums in java
14. Generics in java
15. Linked hashmap internal working
16. arraylist vs linkedlist
17. what is profiler in java explain in simple
18. what are thread dumps in java
19. what is rabbit mq and how to use it. give code in simplest
20. point to point vs pub/sub usage
21. future and completable future in simple.
22. explain executor framework in simplest.
23. garbage collector internal working.
24. Given employees list. use stream and print all employees in a department
25. Given employees list. use stream and print total salary of employees by department.
26. Given employees list. use stream and print total salary of all employees
27. Given employees list. use stream and print highest salary of employees.
28. Given employees list. use stream and print highest salary of employees in each department.

Explain these questions in easy to remember language



Here's a simplified explanation of each of your topics, with code snippets where applicable. I've kept the explanations concise and easy to remember.

1. What is Swagger and How to Use It in Spring Boot

Swagger is a tool that helps you document and test your REST APIs. It provides an interactive interface to explore API endpoints.

How to Use:

1. Add dependencies in `pom.xml`:

xml

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-boot-starter</artifactId>
  <version>3.0.0</version>
</dependency>
```

2. Create a configuration class:

java

```
import springfox.documentation.builders.PathSelectors;
import springfox.documentation.builders.RequestHandlerSelectors;
import springfox.documentation.spi.DocumentationType;
import springfox.documentation.spring.web.plugins.Docket;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors.any())
            .paths(PathSelectors.any())
            .build();
    }
}
```

3. Access Swagger UI at `http://localhost:8080/swagger-ui/`.

2. What is RestTemplate and How to Send Response Using It

RestTemplate is a synchronous client provided by Spring to make HTTP requests.

Code Example:

java

```
import org.springframework.stereotype.Service;
import org.springframework.web.client.RestTemplate;

@Service
public class MyService {
    private final RestTemplate restTemplate;

    public MyService(RestTemplate restTemplate) {
        this.restTemplate = restTemplate;
    }

    public String getResponse(String url) {
        return restTemplate.getForObject(url, String.class);
    }
}
```

3. Connect and Response Time in Spring Boot Microservices

To measure response time in a Spring Boot microservice, you can use **Spring AOP** to log execution time.

Code Example:

```
java

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {
    @Around("execution(* com.example.*(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object proceed = joinPoint.proceed();
        long executionTime = System.currentTimeMillis() - start;
        System.out.println(joinPoint.getSignature() + " executed in " + executionTime +
"ms");
        return proceed;
    }
}
```

4. Local Transaction in Microservices

Local transactions are used when you want to ensure that a specific service's operations are completed successfully. If one operation fails, the other changes are not committed.

You can manage local transactions using Spring's `@Transactional` annotation.

Code Example:

```
java

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
public class UserService {
    @Transactional
    public void createUser(User user) {
        // Save user and related operations
    }
}
```

5. Failures in Microservices and How to Handle

Failures can occur due to network issues, timeouts, or service unavailability. **Resilience** patterns like Circuit Breaker or Retry can help handle these failures.

Using Resilience4j:

1. Add dependency:

xml

```
<dependency>
  <groupId>io.github.resilience4j</groupId>
  <artifactId>resilience4j-spring-boot2</artifactId>
  <version>1.7.1</version>
</dependency>
```

2. Example:

java

```
import io.github.resilience4j.circuitbreaker.annotation.CircuitBreaker;
import org.springframework.stereotype.Service;

@Service
public class ApiService {
    @CircuitBreaker
    public String callExternalService() {
        // Call external service
        return "response";
    }
}
```

6. Service Mesh: What is It?

A **service mesh** is a dedicated infrastructure layer that manages service-to-service communication, providing features like traffic management, security, and observability without requiring changes to the application code.

7. Redis Integration with Spring Boot

Redis is an in-memory data store, often used for caching.

Code Example:

1. Add dependency:

xml

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

2. Configure Redis:

```
java
```

```
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.redis.connection.RedisConnectionFactory;
import org.springframework.data.redis.core.RedisTemplate;

@Configuration
public class RedisConfig {
    @Bean
    public RedisTemplate<String, Object> redisTemplate(RedisConnectionFactory
connectionFactory) {
        RedisTemplate<String, Object> template = new RedisTemplate<>();
        template.setConnectionFactory(connectionFactory);
        return template;
    }
}
```

8. Spring Boot Caching

Spring Boot caching helps improve performance by storing the results of expensive method calls.

Code Example:

1. Enable caching:

```
java
```

```
import org.springframework.cache.annotation.EnableCaching;
import org.springframework.context.annotation.Configuration;

@Configuration
@EnableCaching
public class CacheConfig {
}
```

2. Use caching in a service:

```
java
```

```
import org.springframework.cache.annotation.Cacheable;
import org.springframework.stereotype.Service;

@Service
public class UserService {
    @Cacheable("users")
    public User getUserById(Long id) {
        // Simulate a slow database call
        return fetchUserFromDatabase(id);
    }
}
```

9. How to Deploy Apps on Pivotal Cloud Foundry

1. **Install the CLI** and login to your PCF account.

```
bash
```

```
cf login -a API_ENDPOINT -u USERNAME -p PASSWORD
```

2. Push your app:

```
bash
```

```
cf push APP_NAME -p PATH_TO_YOUR_JAR
```

10. Jenkins Working and How to Create Pipeline

Jenkins is an open-source automation server that helps automate the parts of software development related to building, testing, and deploying.

Creating a Pipeline:

1. Create a new item in Jenkins and select "Pipeline".
2. In the pipeline script, define stages:

```
groovy
```

```
pipeline {
  agent any
  stages {
    stage('Build') {
      steps {
        echo 'Building...'
      }
    }
    stage('Test') {
      steps {
        echo 'Testing...'
      }
    }
    stage('Deploy') {
      steps {
        echo 'Deploying...'
      }
    }
  }
}
```

11. Docker Working and How to Create a Pipeline

Docker is a platform for developing, shipping, and running applications in containers.

Creating a Docker Pipeline:

1. Create Dockerfile:

Dockerfile

```
FROM openjdk:11
COPY target/myapp.jar myapp.jar
ENTRYPOINT ["java", "-jar", "myapp.jar"]
```

2. Build and push Docker image:

bash

```
docker build -t myapp:latest .
docker push myapp:latest
```

3. In Jenkins, create a pipeline to build Docker images.

12. Some Details About Splunk and Splunk Query

Splunk is a data platform that enables you to collect, analyze, and visualize machine-generated data.

Why It's Needed:

- Monitoring logs for troubleshooting.
- Security information and event management (SIEM).

Splunk Query Example:

spl

```
index=myindex sourcetype=access_combined | stats count by status
```

13. Enums in Java

Enums (enumerations) are a special Java type used to define collections of constants.

Code Example:

java

```
public enum Day {
    SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY
}
```

14. Generics in Java

Generics enable types (classes and interfaces) to be parameters when defining classes, interfaces, and methods.

Code Example:

java

```
public class Box<T> {
    private T item;
    public void setItem(T item) {
        this.item = item;
    }
    public T getItem() {
        return item;
    }
}
```

15. Linked HashMap Internal Working

LinkedHashMap maintains a doubly-linked list running through its entries. This allows it to maintain insertion order.

16. ArrayList vs LinkedList

- **ArrayList:**
 - Resizable array.
 - Fast for random access.
 - Slower for insertions/deletions.
- **LinkedList:**
 - Doubly linked list.
 - Faster for insertions/deletions.
 - Slower for random access.

17. What is Profiler in Java

A **profiler** is a tool that measures the space (memory) and time complexity of a program. It helps identify bottlenecks in the code.

18. What are Thread Dumps in Java

Thread dumps provide a snapshot of all threads in a JVM at a given time. They help diagnose deadlocks, performance issues, and application states.

19. What is RabbitMQ and How to Use It

RabbitMQ is an open-source message broker that facilitates communication between services.

Code Example:

1. Add dependencies:

xml

```
<dependency>
  <groupId>org.springframework.boot</groupId>
```



```
<artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

2. Send a message:

```
java
```

```
import org.springframework.amqp.rabbit.core.RabbitTemplate;
import org.springframework.stereotype.Service;

@Service
public class MessagingService {
    private final RabbitTemplate rabbitTemplate;

    public MessagingService(RabbitTemplate rabbitTemplate) {
        this.rabbitTemplate = rabbitTemplate;
    }

    public void sendMessage(String message) {
        rabbitTemplate.convertAndSend("myExchange", "myRoutingKey", message);
    }
}
```

20. Point to Point vs Pub/Sub Usage

- **Point to Point:** One producer and one consumer. Good for direct communication.
- **Pub/Sub:** One producer, multiple consumers. Useful for broadcasting messages.

21. Future and CompletableFuture in Simple

Future is an interface representing the result of an asynchronous computation. **CompletableFuture** allows you to complete it manually.

Example:

```
java
```

```
import java.util.concurrent.CompletableFuture;

CompletableFuture<String> future = CompletableFuture.supplyAsync(() -> {
    return "Hello";
});
future.thenAccept(result -> System.out.println(result));
```

22. Explain Executor Framework in Simplest

The **Executor framework** provides a pool of threads to manage concurrent tasks. It simplifies the task of thread management.

Code Example:

```
java
```

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

ExecutorService executor = Executors.newFixedThreadPool(3);
executor.submit(() -> System.out.println("Task executed"));
executor.shutdown();
```

23. Garbage Collector Internal Working

The **Garbage Collector (GC)** automatically manages memory by removing objects that are no longer referenced, freeing up resources. It runs in the background and has different algorithms like mark-and-sweep, generational GC, etc.

24-28. Given Employees List Using Stream API

Assuming you have a `List<Employee> employees` where `Employee` has `department` and `salary` fields:

24. Print All Employees in a Department:

```
java
```

```
employees.stream()
    .filter(e -> "HR".equals(e.getDepartment()))
    .forEach(System.out::println);
```

25. Print Total Salary of Employees by Department:

```
java
```

```
employees.stream()
    .collect(Collectors.groupingBy(Employee::getDepartment,
        Collectors.summingDouble(Employee::getSalary)))
    .forEach((dept, totalSalary) -> System.out.println(dept + ": " + totalSalary));
```

26. Print Total Salary of All Employees:

```
java
```

```
double totalSalary = employees.stream()
    .mapToDouble(Employee::getSalary)
    .sum();
System.out.println("Total Salary: " + totalSalary);
```

27. Print Highest Salary of Employees:

java

```
employees.stream()
    .mapToDouble(Employee::getSalary)
    .max()
    .ifPresent(maxSalary -> System.out.println("Highest Salary: " + maxSalary));
```

28. Print Highest Salary of Employees in Each Department:

java

```
employees.stream()
    .collect(Collectors.groupingBy(Employee::getDepartment,
        Collectors.mapping(Employee::getSalary,
            Collectors.maxBy(Double::compareTo))))
    .forEach((dept, maxSalary) -> System.out.println(dept + ": " + maxSalary));
```

These simplified explanations and code snippets should give you a good understanding of the topics and help you remember them easily! If you need more details on any specific point, feel free to ask.

ChatGPT can make mistakes. Check important info.