Ali P8 análisis de Spring Boot y Apache Kafka combinados para lograr el manejo de errores, conversión de mensajes y soporte de transacciones?

Etiquetas: Programa de vida Java parte trasera Arquitectura Lenguaje de programación

Spring trae un modelo de programación Spring familiar a Kafka. Proporciona un KafkaTemplate para publicar registros y un contenedor de escucha para ejecutar asincrónicamente oyentes POJO. La configuración automática de Spring Boot conecta la mayor parte de la infraestructura para que pueda concentrarse en la lógica empresarial.

Error de recuperación

Considere este sencillo método de escucha POJO:

```
@KafkaListener(id = "fooGroup", topics = "topic1")
public void listen(String in) {
 logger.info("Received: " + in);
  if (in.startsWith("foo")) {
throw new RuntimeException("failed");
}
```

Por defecto, solo se registran los registros fallidos, y luego pasamos al siguiente registro. Sin embargo, podemos configurar manejadores de errores en el contenedor de escucha para realizar otras operaciones. Para hacer esto, utilizamos nuestros propios métodos para anular la fábrica de contenedores de configuración automática de Spring Boot:

```
                                    @Bean
public ConcurrentKafkaListenerContainerFactory kafkaListenerContainerFactory(
          ConcurrentKafkaListenerContainerFactoryConfigurer configurer,
             ConsumerFactory<Object, Object> kafkaConsumerFactory) {
ConcurrentKafkaListenerContainerFactory<Object, Object> factory = new Concurren
              configurer.configure(factory, kafkaConsumerFactory);
      factory.setErrorHandler(new SeekToCurrentErrorHandler()); // <<<<<<
                                return factory;
                                      }
```

Tenga en cuenta que aún podemos aprovechar la mayoría de la configuración automática.

SeekToCurrentErrorHandler descarta los registros restantes de poll () y realiza una operación de búsqueda para implementar el restablecimiento del desplazamiento del desplazamiento de la operación del consumidor, lo que hace que el registro de descarte se recupere nuevamente en la próxima encuesta. De manera predeterminada, el controlador de errores realiza un seguimiento de los registros fallidos, renuncia y registra registros fallidos después de 10 intentos de entrega. Sin embargo, también podemos enviar un mensaje fallido a otro tema. Lo llamamos el tema de la carta de la muerte.

Aquí está el código juntos:

```
                                    @Bean
public ConcurrentKafkaListenerContainerFactory kafkaListenerContainerFactory(
          ConcurrentKafkaListenerContainerFactoryConfigurer configurer,
              ConsumerFactory<Object, Object> kafkaConsumerFactory,
                    KafkaTemplate<Object, Object> template) {
ConcurrentKafkaListenerContainerFactory<Object, Object> factory = new Concurren
              configurer.configure(factory, kafkaConsumerFactory);
             factory.setErrorHandler(new SeekToCurrentErrorHandler(
                 new DeadLetterPublishingRecoverer(template), 3));
                                return factory;
                                      }

              @KafkaListener(id = "fooGroup", topics = "topic1")
                        public void listen(String in) {
                         logger.info("Received: " + in);
                           if (in.startsWith("foo")) {
                        throw new RuntimeException("failed");
```

```
    }
}

@KafkaListener(id = "dltGroup", topics = "topic1.DLT")
public void dltListen(String in) {
    logger.info("Received from DLT: " + in);
}
```

Error de deserialización

Pero, ¿qué pasa con las excepciones de deserialización que ocurren antes de que Spring obtenga el récord? Utilice ErrorHandlingDeserializer. Este deserializador envuelve el deserializador y atrapa cualquier excepción. Luego se reenvían al contenedor de escucha, que los envía directamente al controlador de errores. Esta excepción contiene datos de origen para que pueda diagnosticar el problema.

Objeto de dominio y tipo inferido

Considere el siguiente ejemplo:

```
@Bean
public RecordMessageConverter converter() {
    return new StringJsonMessageConverter();
}

@KafkaListener(id = "fooGroup", topics = "topic1")
public void listen(Foo2 foo) {
    logger.info("Received: " + foo);
    if (foo.getFoo().startsWith("fail")) {
        throw new RuntimeException("failed");
    }
}

@KafkaListener(id = "dltGroup", topics = "topic1.DLT")
public void dltListen(Foo2 in) {
    logger.info("Received from DLT: " + in);
}
```

Tenga en cuenta que ahora estamos usando el tipo de objeto Foo2. El bean convertidor de mensajes infiere el tipo de tipo de parámetro que se convertirá en una firma de método. El convertidor "confía" automáticamente en este tipo. Spring Boot configura automáticamente el convertidor en el contenedor de escucha.

En el lado del productor, el objeto que se envía puede ser una clase diferente (siempre que sea compatible con el tipo):

```
@RestController
public class Controller {

@Autowired
private KafkaTemplate<Object, Object> template;

@PostMapping(path = "/send/foo/{what}")
public void sendFoo(@PathVariable String what) {
this.template.send("topic1", new Foo1(what));
}

}
```

Configuración:

```
spring:
kafka:
producer:
value-serializer: org.springframework.kafka.support.serializer.JsonSerializ

$ curl -X POST http://localhost:8080/send/foo/fail
```

Aquí usamos el StringDeserializer y un convertidor de mensajes "inteligente" en el lado del consumidor.

Oyente de métodos múltiples

También podemos usar un solo contenedor de escucha y enrutar a un método específico por tipo. Como hay varios métodos, el tipo debe elegir el método al que llamar, por lo que no podemos inferir el tipo aquí.

En cambio, confiamos en la información de tipo que se pasa en el encabezado del registro para asignar del tipo de origen al tipo de destino. Además, dado que no inferimos el tipo, necesitamos configurar el convertidor de mensajes para "confiar" en el tipo de mapeo del paquete.

En este caso, utilizaremos un convertidor de mensajes (StringSerializer y StringDeserializer juntos) en ambos lados. Los siguientes ejemplos de convertidor del lado del consumidor los unen:

```
@Bean
public RecordMessageConverter converter() {
StringJsonMessageConverter converter = new StringJsonMessageConve
```

```
DefaultJackson2JavaTypeMapper typeMapper = new DefaultJackson2JavaTypeMapper();
typeMapper.setTypePrecedence(TypePrecedence.TYPE_ID);
typeMapper.addTrustedPackages("com.common");
Map<String, Class<?>> mappings = new HashMap<>();
mappings.put("foo", Foo2.class);
mappings.put("bar", Bar2.class);
typeMapper.setIdClassMapping(mappings);
converter.setTypeMapper(typeMapper);
return converter;
}
```

Aquí asignamos "foo" a la clase Foo2 y "bar" a la clase Bar2. Tenga en cuenta que debemos decirle que use el encabezado TYPE_ID para determinar el tipo de conversión. Del mismo modo, Spring Boot configura automáticamente el convertidor de mensajes en un contenedor. La siguiente es la asignación de tipo del lado del productor en el fragmento de archivo application.yml; el formato es un token separado por token: lista FQCN:

```
spring:
  kafka:
    producer:
      value-serializer: org.springframework.kafka.support.serializer.JsonSerializ
      properties:
        spring.json.type.mapping: foo:com.common.Foo1,bar:com.common.Bar1
```

Esta configuración asigna la clase Foo1 a "foo" y la clase asigna Bar1 a "bar".

Oyente:

```
@Component
@KafkaListener(id = "multiGroup", topics = { "foos", "bars" })
public class MultiMethods {

@KafkaHandler
public void foo(Foo1 foo) {
System.out.println("Received: " + foo);
}

@KafkaHandler
public void bar(Bar bar) {
System.out.println("Received: " + bar);
}

@KafkaHandler(isDefault = true)
public void unknown(Object object) {
System.out.println("Received unknown: " + object);
```

```
18                                      }
19
20                                  }
```

Productor:

```
1                                   @RestController
2                               public class Controller {
3
4                                   @Autowired
5                           private KafkaTemplate<Object, Object> template;
6
7                               @PostMapping(path = "/send/foo/{what}")
8                           public void sendFoo(@PathVariable String what) {
9                       this.template.send(new GenericMessage<>(new Foo1(what),
10                          Collections.singletonMap(KafkaHeaders.TOPIC, "foos")));
11                                  }
12
13                              @PostMapping(path = "/send/bar/{what}")
14                          public void sendBar(@PathVariable String what) {
15                      this.template.send(new GenericMessage<>(new Bar(what),
16                          Collections.singletonMap(KafkaHeaders.TOPIC, "bars")));
17                                  }
18
19                              @PostMapping(path = "/send/unknown/{what}")
20                      public void sendUnknown(@PathVariable String what) {
21                           this.template.send(new GenericMessage<>(what,
22                          Collections.singletonMap(KafkaHeaders.TOPIC, "bars")));
23                                  }
24
25                                  }
```

Transacción

Habilite las transacciones configurando transaccional-id-prefijo en el archivo application.yml:

```
1                                   spring:
2                                      kafka:
3                                     producer:
4           value-serializer: org.springframework.kafka.support.serializer.JsonSerial
5                               transaction-id-prefix: tx.
6                                      consumer:
7                                      properties:
8                               isolation.level: read_committed
```

Cuando se utiliza spring-kafka 1.3.xo superior y la versión kafka-clients (0.11 o superior) que admite transacciones, cualquier operación @KafkaListener que realice KafkaTemplate en el método participará en la transacción, y el contenedor de escucha será Enviar el compensación a la transacción antes de confirmarla. Me di cuenta de que también establecimos el nivel de aislamiento para los consumidores para que no pudieran ver registros no confirmados. El siguiente ejemplo detiene al oyente para que podamos ver este efecto:

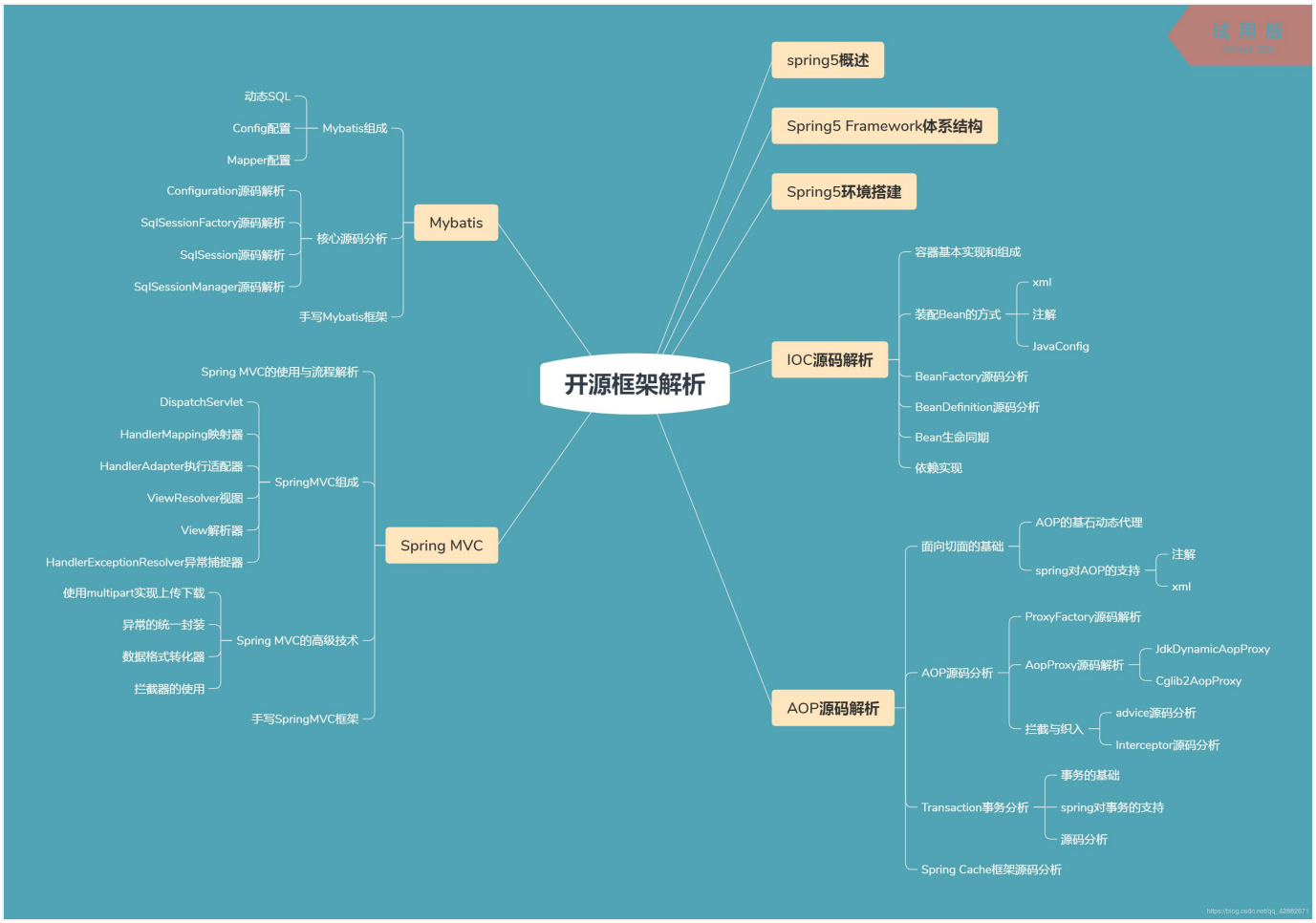
```
@KafkaListener(id = "fooGroup2", topics = "topic2")
 public void listen(List foos) throws IOException {
logger.info("Received: " + foos);
foos.forEach(f -> kafkaTemplate.send("topic3", f.getFoo().toUpperCase()));
logger.info("Messages sent, hit enter to commit tx");
System.in.read();
}

@KafkaListener(id = "fooGroup3", topics = "topic3")
public void listen(String in) {
 logger.info("Received: " + in);
}
```

El productor de este ejemplo envía múltiples registros en una sola transacción:

```
@PostMapping(path = "/send/foos/{what}")
  public void sendFoo(@PathVariable String what) {
this.template.executeInTransaction(kafkaTemplate -> {
  StringUtils.commaDelimitedListToSet(what).stream()
.map(s -> new Foo1(s))
 .forEach(foo -> kafkaTemplate.send("topic2", foo));
return null;
});
}

curl -X POST http://localhost:8080/send/foos/a,b,c,d,e

Received: [Foo2 [foo=a], Foo2 [foo=b], Foo2 [foo=c], Foo2 [foo=d], Foo2 [foo=e]]
Messages sent, hit Enter to commit tx

Received: [A, B, C, D, E]
```

Para los programadores de Java, recomiendo dos temas principales, no solo Spring Boot y Apache Kafka, sino que también incluyen Mybatis, spring cloud, spring MVC, spring boot, spring5, código fuente IOC, código fuente AOP, docker, dubbo y otras tecnologías de arquitectura. Al mismo tiempo, he recopilado algunos materiales de aprendizaje y videos sobre las tecnologías mencionadas en Internet, con la esperanza de ayudar a todos

Análisis de marco de código abierto



Microservicios

Recomendación inteligente

Mecanismo de manejo de errores de arranque de resorte

Mecanismo de manejo de errores Cuando el programa tiene un error Acceso al navegador El inicio de Spring proporciona una página de error predeterminada que incluye el código de estado de error, el tipo de error, el mensaje de solicitud, la hora Acceso al cliente Cuando el programa

---

Manejo de errores de Spring Boot (4)

Desarrollo de aplicaciones web: manejo de errores 1 Método 1: Spring Boot asigna todos los errores a / error de forma predeterminada.ErrorController 2 Método 2: A

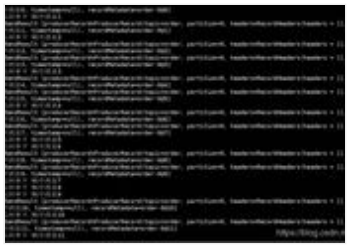
página de error personalizada Página estática HTML: definida en recursos / ...

Mecanismo de manejo de errores de Spring Boot

Resumen del enlace original: 1) El efecto predeterminado del mecanismo de manejo de errores predeterminado de SpringBoot: 1), navegador, devuelve un encabezado de solicitud de página de error predeterminado enviado por el navegador: 2), si es otro cliente, ...

Spring-boot e implemente el transmisor de mensajes kafka

1, la configuración kafakaproducer y consumidor. 2, la devolución de llamada enviando un mensaje de transmisión de manera exitosa o fallida. ...



Mensaje de secuencia de ejemplo de Kafka (arranque de primavera)

springboot kafka sequential message Method to realize:Send messages that need to be consumed sequentially to the same partition ********** Usage exampl...

More Recommendation

Ali P8 architect handwritten notes: Spring source + JVM + MySQL + Kafka + Tomcat

This set of handwritten notes from Ali's senior architect! I believe that many of these knowledge points are worthy of students to learn! If you are interested in the PDF of these notes, please welcom...

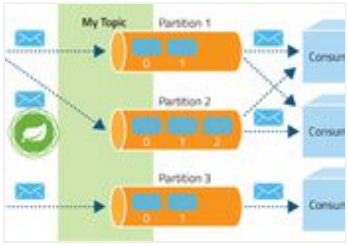
Intimidad

Spring Boot combined with jQuery to achieve selective dele tion

Service layer (Service) Service directly inherits the methods in other classes inherited by DAO, so the DAO layer does not need to add methods. Note: The brackets are the classes inherited by the DAO ...

Ali P8 master masterpiece: Spring source c ode in-depth interpretation (IOC / AOP / M VC / transaction)

For most people looking at the spring source code for the first time, they will feel that they do n't know where to start, because the spring project source code consists of multiple subprojects...



Java Spring Boot 2.0 combat million-level K afka message middleware and principle an alysis

Apache Kafka is an open source distributed high-concurrency messaging middleware that supports millions of messages per second concurrently. It has high concurrency architecture on the Internet: dual ...

spring boot Exception Handling Vulnerability Scanner to sol ve the problem of the application error message

Después del tiempo transcurrido desde el escaneo de vulnerabilidades, las propiedades de los objetos de usuario strName controller se reciben en strName [] = Por lo tanto, se informará una excepción al frente El escaneo de vulnerabilidades ...

Artículos Relacionados

- [Manejo de problemas] Capacidad insuficiente de mensajes de Kafka y resolución de problemas en Spring Boot
- Kafka usa + bota de resorte combinada
- Integración de Spring Boot: conversión de mensaje de devolución de llamada de mensaje de confirmación de configuración de RabitMQ y manejo de excepciones de mensaje
- Manejo de errores de arranque de primavera
- Manejo de errores de Spring Boot
- error de manejo de arranque de primavera
- Manejo de errores de Spring Boot
- Kafka enviar mensaje-transacción-Spring integración
- Episode 3 Spring for Apache Kafka acepta el mensaje
- Principio de manejo de errores de arranque de resorte

entradas populares

- [2] señala que el nuevo autodesarrollado mi primera aplicación de Android
- "El programador pragmático: desde pequeños trabajos hasta las notas de estudio de los expertos (a)
- Primeras imágenes de eco antes de subir
- Atributo peatonal "Aprendizaje débilmente supervisado de características de nivel medio para reconocimiento de atributos peatonales y Loca"
- 1- máquina de vector de soporte SVM linealmente separable con el espacio máximo maximizado
- Diferencias y opciones entre MyISAM e InnoDB, resumen detallado, comparación de rendimiento
- Instalar el servidor mosquitto mqtt en Raspberry Pi Raspberry Pi
- 2. Sistema de archivos

- Dividido
- Comprender los patrones de diseño con el código OC (1) Modo creado

Publicaciones recomendadas

- Inicio del clúster Kafka y secuencia de comandos de detención
- Comunicación y valor del componente principal del niño angular
- Gramática básica de Golang: uso detallado de variables
- JSON
- Cómo cancelar el menú de herramientas que aparece después de que XenDestop5 inicia sesión.
- Parte lógica del algoritmo PhxPaxos (1) -Introducción
- 1.1 Editar elevación del piso
- Tres métodos para configurar variables de entorno de Linux: / etc / profile, ~ / .bashrc, shell
- Aprendizaje profundo y combate TensorFlow (ocho) bases de redes neuronales convolucionales

- Qt dibuja una curva suave

Etiquetas Relacionadas

Bota de primavera

Kafka

consumo

kafka

bota de primavera

manejo de errores de arranque de primavera

Primavera para Apache Kafka

Recibir mensaje

Spring Kafka

Aceptar el mensaje

Intimidad