9

Access to Reactive Stream Data Source

Java Backend Developer I

Objetivos

Comprender los conceptos:

- Access Data Reactive R2
  - SQL Server
  - Mongo
  - Redis
- Spring Cloud Stream
  - Kafka
  - Event Hubs

Agenda

Revisión de los siguientes conceptos:

- Access Data Reactive R2
  - SQL Server
  - Mongo
  - Redis
- Spring Cloud Stream
  - Kafka
  - Event Hubs

Access Data Reactive R2

Introducción

Los sistemas reactivos tienen ciertas características que los hacen ideales para cargas de trabajo de baja latencia y alto rendimiento.

Project Reactor y el portafolio de Spring trabajan juntos para permitir a los desarrolladores crear sistemas reactivos de nivel empresarial que sean receptivos, resistentes, elásticos y basados en mensajes.

Para ello es necesario tener el soporte a conexiones a base de datos de manera reactiva. Spring y R2DBC lo brindan.

Access Data Reactive R2

R2DBC

Concepto

R2DBC significa Conectividad de base de datos relacional reactiva.

R2DBC comenzó como un experimento y una prueba de concepto para permitir la integración de bases de datos relacionales en sistemas que utilizan modelos de programación reactivos: reactivos en el sentido de un modelo de programación funcional, sin bloqueo y controlado por eventos que no hace suposiciones sobre concurrencia o asincronía.

En cambio, supone que la programación y la paralelización suceden como parte de la programación en tiempo de ejecución.

Access Data Reactive R2

R2DBC

Caracteristicas

- R2DBC se basa en la especificación de Reactive Streams, que proporciona una API sin bloqueo totalmente reactiva.
- Trabaja con bases de datos relacionales. A diferencia de la naturaleza bloqueante de JDBC, R2DBC le permite trabajar con bases de datos SQL utilizando una API reactiva.
- Admite soluciones escalables. Con Reactive Streams, R2DBC le permite pasar del modelo clásico a un enfoque más escalable.
- Proporciona una especificación abierta. R2DBC es una especificación abierta y establece una interfaz de proveedor de servicios (SPI) para que los proveedores de controladores implementen y los clientes los consuman.

Access Data Reactive R2

R2DBC

Implementaciones

Actualmente existen las siguientes implementaciones :

- cloud-spanner-r2dbc: controlador para Google Cloud Spanner
- jasync-sql: contenedor R2DBC para Java & Kotlin Async Database Driver para MySQL y PostgreSQL escrito en Kotlin.
- r2dbc-h2: controlador nativo implementado para H2 como base de datos de prueba.
- r2dbc-mariadb: controlador nativo implementado para MariaDB.

Access Data Reactive R2

R2DBC

Implementaciones

- Spring Data R2DBC admite controladores a través del mecanismo SPI conectivity de R2DBC. Puede usar cualquier controlador que implemente la especificación R2DBC con Spring Data R2DBC. Dado que Spring Data R2DBC reacciona a características específicas de cada base de datos, requiere un Dialecto de implementación, de lo contrario, su aplicación no se iniciará. Spring Data R2DBC se envía con implementaciones de dialecto para los siguientes controladores por ejemplo:
- r2dbc-mssql: controlador nativo implementado para Microsoft SQL Server.
- r2dbc-mysql: controlador nativo implementado para MySQL.

Access Data Reactive R2

Reactive SQL Server

Uso y configuración de R2DBC

R2DBC es una especificación para controladores reactivos para Java. RD2BC SQL Server es la implementación para Microsoft SQL Server.

Spring Data R2DBC reacts mediante el ConnectionFactory selecciona el dialecto de base de datos apropiado. En consecuencia R2dbcDialect debe configurar el suyo si Spring Data R2DBC aún no conoce el controlador que usa.

Access Data Reactive R2

Reactive SQL Server

Uso y configuración de R2DBC

Los pre-requisitos para utilizar Spring Data R2DBC con Azure SQL database:

- Una cuenta de Azure. Si no tiene uno, obtenga una prueba gratuita .
- Azure Cloud Shell o CLI de Azure . Recomendamos Azure Cloud Shell para que inicie sesión automáticamente y tenga acceso a todas las herramientas que necesitará.
- Un kit de desarrollo de Java compatible , versión 8 (incluido en Azure Cloud Shell).
- cURL o una utilidad HTTP similar para probar la funcionalidad.

Access Data Reactive R2

Reactive MongoDB

Caracteristicas

El soporte reactivo de MongoDB contiene el siguiente conjunto básico de características:

- Soporte de configuración de Spring que utiliza **@Configuration** clases basadas en Java, una **MongoClient** instancia y conjuntos de réplicas.
- **ReactiveMongoTemplate**, es una clase auxiliar que aumenta la productividad al usar **MongoOperations** de manera reactiva. Incluye asignación de objetos integrada entre **Document** instancias y POJO.
- Traducción de excepciones a la jerarquía de excepciones de acceso a datos portátil de Spring.

Access Data Reactive R2

Reactive MongoDB

Caracteristicas

El soporte reactivo de MongoDB contiene el siguiente conjunto básico de características:

- Mapeo de objetos reach en características integrado con Spring's **ConversionService**.
- Metadatos basados en anotaciones que son extensibles para admitir otros formatos de metadatos.
- Persistencia y mapeo de eventos del ciclo de vida.
- Basado en Java **Query, Criteriay UpdateDSL.**
- Implementación automática de interfaces de repositorio reactivas que incluyen soporte para métodos de consulta personalizados.

Access Data Reactive R2

Reactive MongoDB

Configuraciones necesarias

- El soporte de Spring MongoDB requiere MongoDB 2.6 o superior y Java SE 8 o superior.
- Primero, debe configurar un servidor MongoDB en ejecución. Consulte la guía de inicio rápido de MongoDB para obtener una explicación sobre cómo iniciar una instancia de MongoDB. Una vez instalado, iniciar MongoDB normalmente es cuestión de ejecutar el siguiente comando:

  $ {MONGO_HOME} / bin / mongod

Access Data Reactive R2

Reactive MongoDB

Empezando con Reactive

- Puede crear un proyecto de Spring, con un nombre de paquete, como por ejemplo org.spring.mongodb.example.
- Luego agregue lo siguiente a la sección de dependencia pom.xml.

```
<dependency>
  <groupId>org.springframework.data</groupId>
  <artifactId>spring-data-mongodb</artifactId>
  <version>3.0.2.RELEASE</version>
</dependency>
```

```
<dependency>
  <groupId>org.mongodb</groupId>
  <artifactId>mongodb-driver-reactivestreams</artifactId>
  <version>4.0.5</version>
</dependency>

<dependency>
  <groupId>io.projectreactor</groupId>
  <artifactId>reactor-core</artifactId>
  <version>Dysprosium-SR10</version>
</dependency>
```

Access Data Reactive R2

Reactive MongoDB

Empezando con Reactive

- Para comenzar con un ejemplo práctico, cree una clase Person simple para persistir, de la siguiente manera (implementamos sus métodos get y set

```
@Document
public class Person {

  private String id;
  private String name;
  private int age;

  public Person(String name, int age) {
    this.name = name;
    this.age = age;
  }
```

Access Data Reactive R2

Reactive MongoDB

Empezando con Reactive

- Luego cree una aplicación para ejecutar, de la siguiente manera:

```
public class ReactiveMongoApp {

  private static final Logger log = LoggerFactory.getLogger(ReactiveMongoApp.class);

  public static void main(String[] args) throws Exception {

    CountDownLatch latch = new CountDownLatch(1);

    ReactiveMongoTemplate mongoOps = new ReactiveMongoTemplate(MongoClients.create(), "database");

    mongoOps.insert(new Person("Joe", 34))
          .flatMap(p -> mongoOps.findOne(new Query(where("name").is("Joe")), Person.class))
          .doOnNext(person -> log.info(person.toString()))
          .flatMap(person -> mongoOps.dropCollection("person"))
          .doOnComplete(latch::countDown)
          .subscribe();

    latch.await();
  }
}
```

Access Data Reactive R2

Reactive MongoDB

Empezando con Reactive

- **En el ejemplo anterior, hay cosas a tener en cuenta:**
- Puede crear una instancia de la clase auxiliar central de Spring Mongo (**ReactiveMongoTemplate**) utilizando el objeto **com.mongodb.reactivestreams.client.MongoClient** estándar y el nombre de la base de datos a utilizar.
- El mapeador funciona con objetos **POJO** estándar sin la necesidad de metadatos adicionales (aunque opcionalmente puede proporcionar esa información.
- Se utilizan convenciones para manejar el campo ID, convirtiéndolo en un **ObjectId** cuando se almacena en la base de datos.

Access Data Reactive R2

Reactive MongoDB

Empezando con Reactive

- Las convenciones de mapeo pueden utilizar el acceso al campo. Observe que la clase **Person** solo tiene captadores.
- Si los nombres de los argumentos del constructor coinciden con los nombres de campo del documento almacenado, se utilizan para crear una instancia del objeto.

Access Data Reactive R2

Reactive MongoDB

Clase ReactiveMongoTemplate

- La clase ReactiveMongoTemplate, ubicada en el paquete org.springframework.data.mongodb, es la clase central del soporte Reactive MongoDB de Spring y proporciona un conjunto de características para interactuar con la base de datos. La plantilla ofrece operaciones para crear, actualizar, eliminar y consultar documentos MongoDB y proporciona un mapeo entre los objetos de su dominio y los documentos MongoDB.
- El mapeo entre los documentos de MongoDB y las clases de dominio se realiza delegando en una implementación de la interfaz **MongoConverter**.
- Spring proporciona una implementación predeterminada con **MongoMappingConverter**, pero también puede escribir su propio convertidor.

Access Data Reactive R2

Reactive MongoDB

Clase ReactiveMongoTemplate

- La clase **ReactiveMongoTemplate** implementa la interfaz **ReactiveMongoOperations**.
- En la medida de lo posible, los métodos en **ReactiveMongoOperations** reflejan los métodos disponibles en el objeto **Collection** de controladores de MongoDB, para que la API sea familiar para los desarrolladores de MongoDB existentes que están acostumbrados a la API de controladores.
- Por ejemplo, puede encontrar métodos como **find, findAndModify, findOne, insert, remove, save, update y updateMulti.**

Access Data Reactive R2

Reactive MongoDB

Clase ReactiveMongoTemplate

- El objetivo del diseño es facilitar al máximo la transición entre el uso del controlador base MongoDB y **ReactiveMongoOperations**. Una diferencia importante entre las dos API es que **ReactiveMongoOperations** puede pasar objetos de dominio en lugar de **Document**, y hay API fluidas para operaciones de **Query**, **Criteria** y **Update** en lugar de completar un documento para especificar los parámetros de esas operaciones.
- La implementación del convertidor predeterminada utilizada por **ReactiveMongoTemplate** es **MappingMongoConverter.** Si bien MappingMongoConverter puede usar metadatos adicionales para especificar la asignación de objetos a documentos, también puede convertir objetos que no contienen metadatos adicionales mediante el uso de algunas convenciones para la asignación de ID y nombres de colección.

Access Data Reactive R2

Reactive MongoDB Repositories

- El universo reactivo ofrece varias bibliotecas de composiciones reactivas. Las bibliotecas más comunes son RxJava y Project Reactor.
- Spring Data MongoDB se basa en el controlador MongoDB Reactive Streams, para proporcionar la máxima interoperabilidad al confiar en la iniciativa Reactive Streams. Las API estáticas, como **ReactiveMongoOperations**, se proporcionan mediante los tipos **Flux y Mono de Project Reactor**. Project Reactor ofrece varios adaptadores para convertir tipos de envoltorios reactivos (**Flux a Observable** y viceversa), pero la conversión puede saturar fácilmente su código.

Access Data Reactive R2

Reactive MongoDB Repositories

- La abstracción del repositorio de Spring Data es una API dinámica, Los repositorios reactivos de MongoDB se pueden implementar utilizando los tipos de contenedor RxJava o Project Reactor extendiéndose desde una de las siguientes interfaces de repositorio específicas de la biblioteca:
  - ReactiveCrudRepository
  - ReactiveSortingRepository
  - RxJava2CrudRepository
  - RxJava2SortingRepository

Access Data Reactive R2

Reactive MongoDB Repositories

Caracteristicas

- Reactive MongoDB de Spring Data viene con un conjunto de funciones reducido en comparación con los repositorios de bloqueo de MongoDB.
- Es compatible con las siguientes funciones:
  - Métodos de consulta que utilizan consultas de cadenas y derivación de consultas
  - Consultas de repositorios geoespaciales
  - Consultas de eliminación de repositorio
  - Métodos de consulta y restricción de campos basados en MongoDB JSON
  - Consultas de búsqueda de texto completo
  - Métodos de consulta con seguridad de tipos
  - Proyecciones

Access Data Reactive R2

Reactive MongoDB Repositories

Caracteristicas

- Reactive MongoDB de Spring Data viene con un conjunto de funciones reducido en comparación con los repositorios de bloqueo de MongoDB.
- Es compatible con las siguientes funciones:
  - Métodos de consulta que utilizan consultas de cadenas y derivación de consultas
  - Consultas de repositorios geoespaciales
  - Consultas de eliminación de repositorio
  - Métodos de consulta y restricción de campos basados en MongoDB JSON
  - Consultas de búsqueda de texto completo
  - Métodos de consulta con seguridad de tipos
  - Proyecciones

Access Data Reactive R2

Reactive MongoDB Repositories

Aplicacion

- Para acceder a las entidades de dominio que esten en una base de datos de MongoDB, puede utilizar el soporte de repositories que facilita su implementación.
- Para hacerlo, se debe crear una interfaz similar para su repositorio. Sin embargo, antes de poder hacer ello, se necesita una entidad, como la entidad definida en el siguiente ejemplo:

```
public class Person {

  @Id
  private String id;
  private String firstname;
  private String lastname;
  private Address address;

  // … getters and setters omitted
}
```

Access Data Reactive R2

Reactive MongoDB Repositories

Aplicacion

- Tenga en cuenta que la entidad definida en el ejemplo anterior tiene una propiedad denominada id de tipo String. El mecanismo de serialización predeterminado utilizado en MongoTemplate (que respalda el soporte del repositorio) considera las propiedades denominadas id como el ID del documento. Actualmente, admitimos String, ObjectId y BigInteger como tipos de identificación. Consulte Mapeo de ID para obtener más información sobre cómo se maneja el campo de ID en la capa de mapeo.

Access Data Reactive R2

Reactive MongoDB Repositories

Aplicacion

- El siguiente ejemplo muestra cómo crear una interfaz que defina consultas contra el objeto Person del ejemplo anterior:

```
public interface ReactivePersonRepository extends ReactiveSortingRepository<Person, String> {

  Flux<Person> findByFirstname(String firstname);                                    1

  Flux<Person> findByFirstname(Publisher<String> firstname);                         2

  Flux<Person> findByFirstnameOrderByLastname(String firstname, Pageable pageable);  3

  Mono<Person> findByFirstnameAndLastname(String firstname, String lastname);        4

  Mono<Person> findFirstByLastname(String lastname);                                 5
}
```

Access Data Reactive R2

Reactive MongoDB Repositories

Explicacion del Codigo Anterior

1. El método muestra una consulta para todas las personas con el apellido dado. La consulta se deriva analizando el nombre del método en busca de restricciones que se pueden concatenar con And y Or. Por lo tanto, el nombre del método da como resultado una expresión de consulta de {"apellido": apellido}.
2. El método muestra una consulta para todas las personas con el nombre dado una vez que el editor dado emite el nombre.
3. Utiliza **Pageable** para pasar parámetros de clasificación y desplazamiento a la base de datos.

Access Data Reactive R2

Reactive MongoDB Repositories

Explicacion del Codigo Anterior

4. Encuentre una sola entidad para los criterios dados. Se completa con una excepción **IncorrectResultSizeDataAccessException** en resultados no únicos.
5. A menos que <4>, la primera entidad siempre se emite incluso si la consulta arroja más documentos de resultado.

Access Data Reactive R2

Reactive Redis

Requerimientos

- Spring Data Redis se integra actualmente con **Lettuce** como el único conector Java reactivo. Project Reactor se utiliza como biblioteca de composición reactiva.

Conexión a Redis mediante un controlador reactivo

- Una de las primeras tareas al usar Redis y Spring es conectarse al store a través del contenedor de IoC. Para hacer eso, se requiere un conector Java (o enlace). Independientemente de la biblioteca que elija, debe usar el paquete **org.springframework.data.redis.connection** y sus interfaces **ReactiveRedisConnection**, **ReactiveRedisConnectionFactory** para trabajar y recuperar conexiones activas a Redis.

Access Data Reactive R2

Reactive Redis

Configuración de un conector Lettuce

- Lettuce es compatible con Spring Data Redis a través del paquete **org.springframework.data.redis.connection.lettuce**.
- Puede configurar **ReactiveRedisConnectionFactory** para Lettuce de la siguiente manera:

```
@Bean
public ReactiveRedisConnectionFactory connectionFactory() {
  return new LettuceConnectionFactory("localhost", 6379);
}
```

Access Data Reactive R2

Reactive Redis

Configuración de un conector Lettuce

- Lettuce es compatible con Spring Data Redis a través del paquete **org.springframework.data.redis.connection.lettuce**.
- Puede configurar **ReactiveRedisConnectionFactory** para Lettuce de la siguiente manera:

Access Data Reactive R2

Reactive Redis

Trabajar con objetos a través de ReactiveRedisTemplate

- Es probable que la mayoría de los usuarios utilicen **ReactiveRedisTemplate** y su paquete correspondiente, **org.springframework.data.redis.core**. el template es, de hecho, la clase central del módulo de Redis. Ofrece una abstracción de alto nivel para las interacciones de Redis.
- Si bien **ReactiveRedisConnection** ofrece métodos de bajo nivel que aceptan y devuelven valores binarios (ByteBuffer), el **template** se encarga de la serialización y la administración de conexiones, lo que le libera de tener que lidiar con esos detalles.
- Además, el template proporciona vistas de operaciones (siguiendo la agrupación de la referencia de comandos de Redis) que ofrecen interfaces enriquecidas.

Access Data Reactive R2

Reactive Redis

Trabajar con objetos a través de ReactiveRedisTemplate

- **ReactiveRedisTemplate** utiliza un serializador basado en Java para la mayoría de sus operaciones. Esto significa que cualquier objeto escrito o leído por la plantilla se serializa o deserializa a través de **RedisElementWriter** o **RedisElementReader**.
- El contexto de serialización se pasa a la plantilla durante la construcción, y el módulo Redis ofrece varias implementaciones disponibles.
  - Todas ellas en el paquete: org.springframework.data.redis.serializer.
- Consulte Serializadores para obtener más información en la pagina oficial de Spring.

Spring Cloud Stream

Introducción

- Spring Cloud Stream es un framework para crear aplicaciones de microservicio basadas en mensajes.
- Spring Cloud Stream se basa en Spring Boot para crear aplicaciones Spring independientes de nivel de producción y utiliza Spring Integration para proporcionar conectividad a los agentes de mensajes.
- Proporciona una configuración de middleware de varios proveedores, presentando los conceptos de semántica de publicación-suscripción persistente, grupos de consumidores y particiones.

Spring Cloud Stream

Introducción

- Permite al agregar la anotación @EnableBinding a la aplicación obtener conectividad inmediata con un agente de mensajes, y puede agregar @StreamListener a un método para hacer que reciba eventos para el procesamiento de transmisión.
- El siguiente ejemplo muestra una aplicación de receptor que recibe mensajes externos:

```
@SpringBootApplication
@EnableBinding(Sink.class)
public class VoteRecordingSinkApplication {

  public static void main(String[] args) {
    SpringApplication.run(VoteRecordingSinkApplication.class, args);
  }

  @StreamListener(Sink.INPUT)
  public void processVote(Vote vote) {
      votingService.recordVote(vote);
  }
}
```

Spring Cloud Stream

Introducción

- La anotación @EnableBinding toma una o más interfaces como parámetros (en este caso, el parámetro es una única interfaz Sink). Una interfaz declara canales de entrada y salida. Spring Cloud Stream proporciona las interfaces de origen, receptor y procesador. También puede definir sus propias interfaces
- La siguiente lista muestra la definición de la interfaz Sink:

```
public interface Sink {
  String INPUT = "input";

  @Input(Sink.INPUT)
  SubscribableChannel input();
}
```

Spring Cloud Stream

Introducción

- La anotación **@Input** identifica un canal de entrada, a través del cual los mensajes recibidos ingresan a la aplicación. La anotación **@Output** identifica un canal de salida, a través del cual los mensajes publicados abandonan la aplicación. Las anotaciones **@Input y @Output** pueden tomar un nombre de canal como parámetro. Si no se proporciona un nombre, se utiliza el nombre del método anotado.
- Spring Cloud Stream crea una implementación de la interfaz para usted.

Spring Cloud Stream

Introducción

- Puede usar ello en la aplicación conectándolo automáticamente, como se muestra en el siguiente ejemplo (de un caso de prueba):

```
@RunWith(SpringJUnit4ClassRunner.class)
@SpringApplicationConfiguration(classes = VoteRecordingSinkApplication.class)
@WebAppConfiguration
@DirtiesContext
public class StreamApplicationTests {

  @Autowired
  private Sink sink;

  @Test
  public void contextLoads() {
    assertThat(this.sink.input()).isNotNull();
  }
}
```

Spring Cloud Stream

KAFKA

- Spring Cloud Stream es una tecnología que nos permite procesar eventos y transacciones en aplicaciones. Agregar la capacidad de interactuar con muchas interfaces de transmisión diferentes permite que Spring Cloud Stream se adapte a las nuevas interfaces del sistema y las nuevas tecnologías de terceros, como Kafka Message Broker.
- Kafka es conocido por poder manejar grandes cantidades de transacciones por minuto. Hay muchos usos para este alto nivel de transacciones y Kafka ha visto un aumento reciente en la adopción en muchas organizaciones.

Spring Cloud Stream

KAFKA

- El puente entre un sistema de mensajería y Spring Cloud Stream es a través de la abstracción de la carpeta. Existen Binders para varios sistemas de mensajería, pero una de las Binders más utilizadas es Apache Kafka.
- Kafka Binder se extiende sobre los sólidos cimientos de Spring Boot, Spring para Apache Kafka y Spring Integration. Dado que el enlazador es una abstracción, también hay implementaciones disponibles para otros sistemas de mensajería.
- Spring Cloud Stream admite semántica de pub / sub, grupos de consumidores y particiones nativas, y delega estas responsabilidades al sistema de mensajería siempre que sea posible.

Spring Cloud Stream

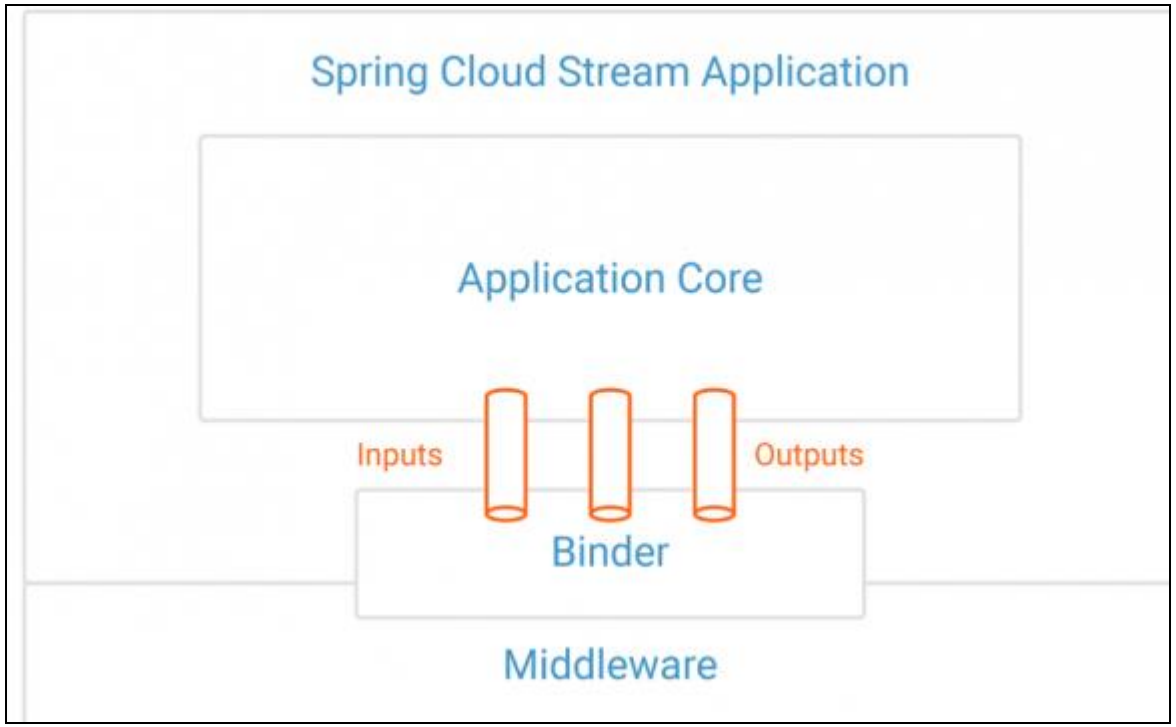
KAFKA

- En el caso del Kafka binder, estos conceptos se asignan internamente y se delegan a Kafka, ya que Kafka los admite de forma nativa. Cuando los sistemas de mensajería no admiten estos conceptos de forma nativa, Spring Cloud Stream los proporciona como características principales.

Spring Cloud Stream

KAFKA

- A continuación, se muestra una representación gráfica de cómo funciona el binder abstraction con entradas y salidas:



Spring Cloud Stream

Descripcion general del soporte con KAFKA

- Cuando se trata de escribir aplicaciones de procesamiento de flujos, Spring Cloud Stream proporciona otro binder específicamente dedicado para Kafka Streams.
- Al igual que con Kafka binder regular, Kafka binder Streams también se centra en la productividad del desarrollador, por lo que los desarrolladores pueden centrarse en escribir la lógica empresarial para KStream, KTable, GlobalKTable, etc., en lugar del código de infraestructura.
- La Binder se encarga de conectarse a Kafka, así como de crear, configurar y mantener las transmisiones y los temas. Por ejemplo, si el método de aplicación tiene una firma de KStream, el binder se conectará al tema de destino y lo transmitirá detrás de escena.

Spring Cloud Stream

Descripcion general del soporte con KAFKA

- El desarrollador de la aplicación no tiene que hacer eso explícitamente, ya que el binder ya lo proporciona para la aplicación.
- Lo mismo se aplica a otros tipos como KTable y GlobalKTable. El enlazador proporciona el objeto subyacente de KafkaStreams para la inyección de dependencia y, por lo tanto, la aplicación no lo mantiene directamente. Más bien, Spring Cloud Stream lo hace por nosotros.

Spring Cloud Stream

Descripcion general del soporte con KAFKA

- Kafka Streams proporciona primitivas de primera clase para escribir aplicaciones con estado. Cuando se crean aplicaciones con estado utilizando Spring Cloud Stream y Kafka Streams, es posible tener aplicaciones RESTful que pueden extraer información de las tiendas de estado persistentes en RocksDB.

Spring Cloud Stream

Descripcion general del soporte con KAFKA

A continuación un ejemplo de una aplicación Spring REST que se basa en las store de Kafka Streams:

```
@RestController
public class FooController {

  private final Log logger = LogFactory.getLog(getClass());

  @Autowired
  private InteractiveQueryService interactiveQueryService;

@RequestMapping("/song/id")
public SongBean song(@RequestParam(value="id") Long id) {

     final ReadOnlyKeyValueStore<Long, Song> songStore =
           interactiveQueryService.getQueryableStore("STORE-NAME",
 QueryableStoreTypes.<Long, Song>keyValueStore());

     final Song song = songStore.get(id);
     if (song == null) {
        throw new IllegalArgumentException("Song not found.");
     }
     return new SongBean(song.getArtist(), song.getAlbum(), song.getName());
  }
}
```

Spring Cloud Stream

Even Hubs

- Spring Cloud Stream Binder para Azure Event Hubs ahora está disponible de forma generalizada.
- Es sencillo crear aplicaciones Java altamente escalables basadas en eventos utilizando Spring Cloud Stream con Event Hubs, un servicio de ingesta de datos en tiempo real totalmente administrado en Azure que es un servicio resistente y confiable para cualquier situación.
- Esto incluye emergencias, gracias a sus funciones de recuperación ante desastres y replicación geográfica.

Spring Cloud Stream

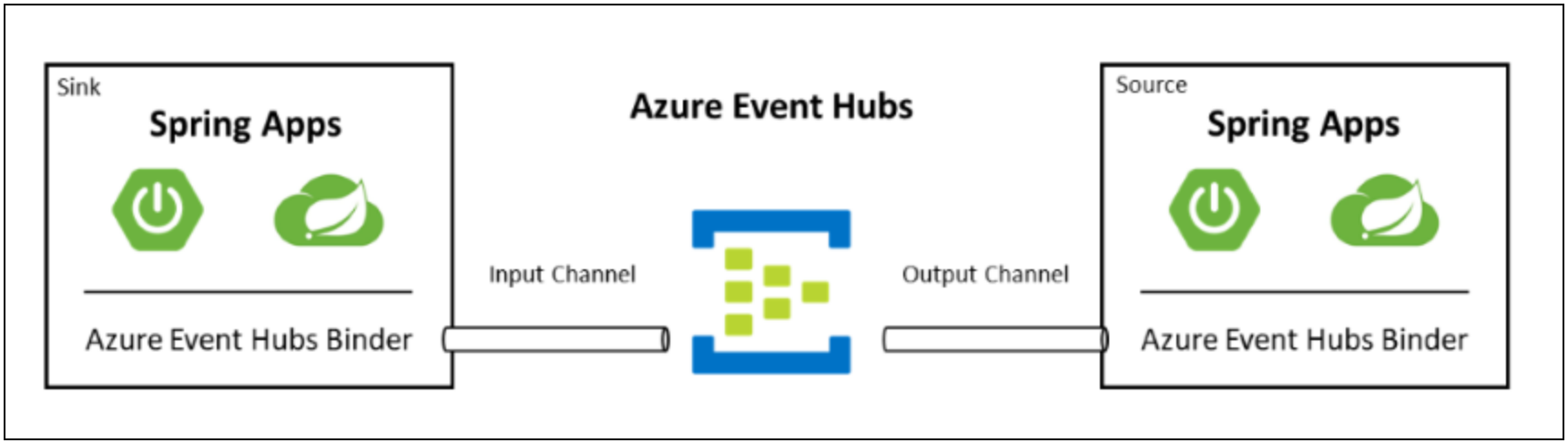
Even Hubs

- Spring Cloud Stream proporciona un binder abstraction para implementaciones de agentes de mensajes (message broker).
- Proporciona un modelo de programación flexible basado en mejores prácticas de Spring ya establecidos y familiares, incluido el soporte para semántica de pub / sub persistente, grupos de consumidores y particiones con estado. Ahora, los desarrolladores pueden usar los mismos patrones para crear aplicaciones Java con Event Hubs.

Spring Cloud Stream

Even Hubs

Diagrama



Spring Cloud Stream

Even Hubs

- En el Azure Portal puede crear un nuevo namespace de Event Hubs. Agregue la siguiente dependencia de Maven a su proyecto Java.

```
<dependency>
    <groupId>com.microsoft.azure</groupId>
    <artifactId>spring-cloud-azure-eventhubs-stream-binder</artifactId>
    <version>1.1.0.RC4</version>
</dependency>
```

Spring Cloud Stream

Even Hubs

Publica mensajes

- Usar @EnableBinding (Source.class) para anotar una clase de origen y publicar mensajes en Event Hubs con patrones Spring Cloud Stream. Puede personalizar el canal de salida para la Fuente con configuraciones.
- Destino: especifique qué centro de eventos conectar con el canal de salida.
- Sync / Async: especifique el modo para producir los mensajes.

Spring Cloud Stream

Even Hubs

Suscribirse a mensajes

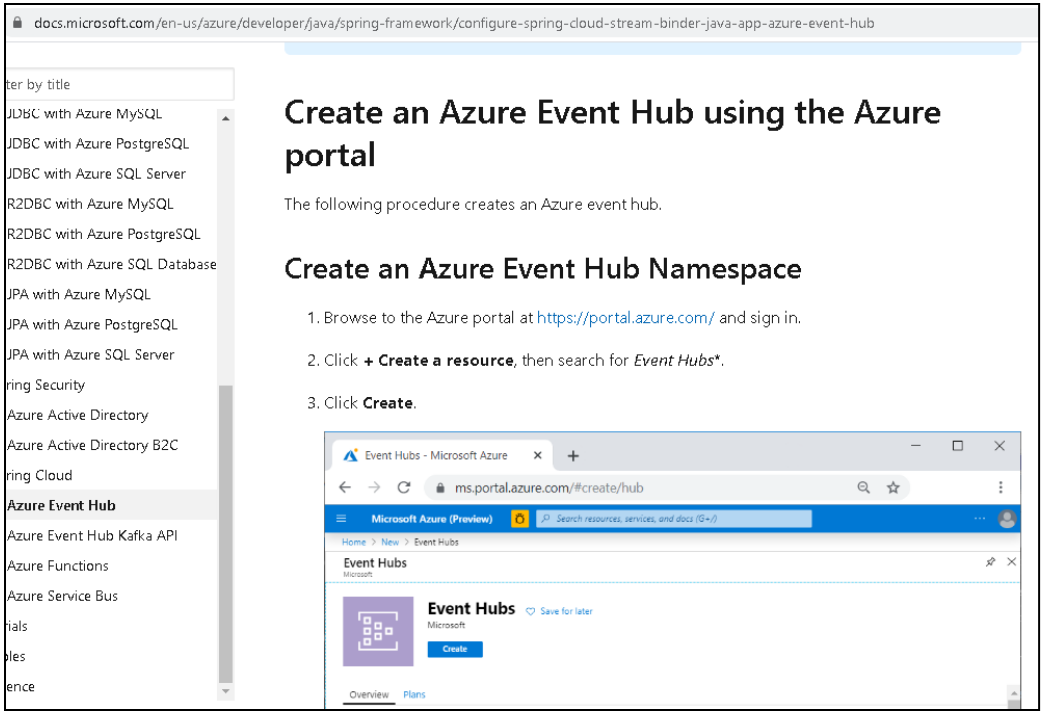
- Use @EnableBinding (Sink.class) para anotar una clase de receptor y consumir mensajes de Event Hubs. También puede personalizar el canal de entrada con configuraciones. Para obtener la lista completa, consulte la documentación, "Cómo crear una aplicación Spring Cloud Stream Binder con Azure Event Hubs".
- Destino: especifique un centro de eventos para enlazar con el canal de entrada.
- Grupo de clientes: especifique un grupo de consumidores para recibir mensajes.

Spring Cloud Stream

Even Hubs

Para mayor información puede ir a la pagina de azure de Microsoft y ver un ejemplo para configurar Even hubs con Spring Cloud Stream:

https://docs.microsoft.com/en-us/azure/developer/java/spring-framework/configure-spring-cloud-stream-binder-java-app-azure-event-hub



Lecturas adicionales

Para obtener información adicional, puede consultar los siguientes enlaces:

- https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.reactive
- https://docs.spring.io/spring-data/mongodb/docs/current/reference/html/#mongo.reactive.repositories
- https://docs.spring.io/spring-data/redis/docs/2.4.0-M1/reference/html/#redis:reactive
- https://cloud.spring.io/spring-cloud-static/spring-cloud-stream/2.2.1.RELEASE/spring-cloud-stream.html#spring-cloud-stream-reference

Resumen

En este capítulo, usted aprendió:

- Access Data Reactive R2
  - SQL Server
  - Mongo
  - Redis
- Spring Cloud Stream
  - Kafka
  - Event Hubs