Manipulação de erros

Conforme descrito na visão geral no início deste manual, uma das principais motivações por trás de uma estrutura orientada a mensagens, como Spring Integration, é promover o acoplamento fraco entre os componentes. O canal de mensagem desempenha um papel importante, na medida em que produtores e consumidores não precisam se conhecer. No entanto, as vantagens também apresentam algumas desvantagens. Algumas coisas se tornam mais complicadas em um ambiente fracamente acoplado, e um exemplo é o tratamento de erros.

Ao enviar uma mensagem a um canal, o componente que lida com essa mensagem pode ou não estar operando no mesmo encadeamento que o remetente. Se estiver usando um padrão simples `DirectChannel` (quando o `<channel>` elemento não possui nenhum `<queue>` elemento filho e nenhum atributo 'task-executor'), o tratamento da mensagem ocorre no mesmo thread que envia a mensagem inicial. Nesse caso, se um `Exception` for lançado, ele pode ser capturado pelo remetente (ou pode se propagar para além do remetente se não for capturado `RuntimeException`). Este é o mesmo comportamento de uma operação de lançamento de exceção em uma pilha de chamadas Java normal.

Um fluxo de mensagens executado em um encadeamento do responsável pela chamada pode ser chamado por meio de um gateway de sistema de mensagens (consulte Gateways do Sistema de Mensagens) ou um `MessagingTemplate` (consulte Recursos `MessagingTemplate`). Em ambos os casos, o comportamento padrão é lançar quaisquer exceções para o chamador. Para o gateway de mensagens, consulte Tratamento de erros para obter detalhes sobre como a exceção é lançada e como configurar o gateway para rotear os erros para um canal de erro. Ao usar um `MessagingTemplate` ou enviar para um `MessageChannel` diretamente, as exceções são sempre lançadas para o chamador.

Ao adicionar processamento assíncrono, as coisas se tornam um pouco mais complicadas. Por exemplo, se o elemento 'canal' fornece um elemento filho 'fila' (`QueueChannel` em Java e Configuração de Anotações), o componente que trata a mensagem opera em um thread diferente do remetente. O mesmo é verdade quando um `ExecutorChannel` é usado. O remetente pode ter caído `Message` no canal e mudado para outras coisas. Não há como `Exception` ser jogado diretamente de volta para aquele remetente usando `Exception` técnicas de lançamento padrão . Em vez disso, o tratamento de erros para processos assíncronos requer que o mecanismo de tratamento de erros também seja assíncrono.

O Spring Integration suporta tratamento de erros para seus componentes publicando erros em um canal de mensagem. Especificamente, `Exception` torna - se a carga útil de uma integração Spring `ErrorMessage` . Isso `Message` é então enviado para um canal de mensagem que é resolvido de forma semelhante à resolução 'replyChannel'. Primeiro, se a solicitação que `Message` está sendo tratada no momento em que `Exception` ocorreu contiver um cabeçalho 'errorChannel' (o nome do cabeçalho é

definido na `MessageHeaders.ERROR_CHANNEL` constante), o `ErrorMessage` é enviado para esse canal. Caso contrário, o manipulador de erros envia para um canal “global” cujo nome do bean é `errorChannel` (isso também é definido como uma constante:) `IntegrationContextUtils.ERROR_CHANNEL_BEAN_NAME`.

Um `errorChannel` bean padrão é criado internamente pelo Framework. No entanto, você pode definir o seu próprio se quiser controlar as configurações. O exemplo a seguir mostra como definir um canal de erro na configuração XML apoiada por uma fila com capacidade de `500`:

```
<int:channel id="errorChannel">
    <int:queue capacity="500"/>
</int:channel>
```

O canal de erro padrão é a `PublishSubscribeChannel`.

A coisa mais importante para entender aqui é que o tratamento de erros baseada em mensagens aplica-se apenas a exceções que são geradas por uma tarefa Spring Integration que está em execução dentro de um `TaskExecutor`. Isso não se aplica a exceções lançadas por um manipulador que opera no mesmo encadeamento que o remetente (por exemplo, por meio de um `DirectChannel` conforme descrito anteriormente nesta seção).

Quando ocorrem exceções na execução de uma tarefa de poller agendada, essas exceções são `ErrorMessage` agrupadas em instâncias e enviadas para o 'errorChannel' também.

Para habilitar o tratamento de erros global, registre um manipulador nesse canal. Por exemplo, você pode configurar o Spring Integration's `ErrorMessageExceptionTypeRouter` como o manipulador de um endpoint que está inscrito no 'errorChannel'. Esse roteador pode então espalhar as mensagens de erro por vários canais, com base no `Exception` tipo.

A partir da versão 4.3.10, o Spring Integration fornece o `ErrorMessagePublisher` e o `ErrorMessageStrategy`. Você pode usá-los como um mecanismo geral para `ErrorMessage` instâncias de publicação . Você pode chamá-los ou estendê-los em qualquer cenário de tratamento de erros. O `ErrorMessageSendingRecoverer` estende essa classe como uma `RecoveryCallback` implementação que pode ser usada com novas tentativas, como o `RequestHandlerRetryAdvice`. O `ErrorMessageStrategy` é usado para construir um `ErrorMessage` baseado na exceção fornecida e um `AttributeAccessor` contexto. Pode ser injetado

em qualquer `MessageProducerSupport` ou `MessagingGatewaySupport`. O `requestMessage` é armazenado em `ErrorMessageUtils.INPUT_MESSAGE_CONTEXT_KEY` no `AttributeAccessor` contexto. Eles `ErrorMessageStrategy` podem usar isso `requestMessage` como `originalMessage` propriedade do `ErrorMessage` que cria. O `DefaultErrorMessageStrategy` faz exatamente isso.

A partir da versão 5.2, todas as `MessageHandlingException` instâncias lançadas pelos componentes da estrutura incluem um `BeanDefinition` recurso de componente e uma fonte para determinar um ponto de configuração da exceção. No caso de configuração XML, um recurso é um caminho de arquivo XML e origina uma tag XML com seu `id` atributo. Com a configuração Java e Annotation, um recurso é uma `@Configuration` classe e uma fonte é um `@Bean` método. Na maioria dos casos, a solução de fluxo de integração de destino é baseada nos componentes prontos para uso e em suas opções de configuração. Quando ocorre uma exceção em tempo de execução, não há nenhum código de usuário final envolvido no rastreamento de pilha porque uma execução é contra os beans, não sua configuração. Incluir um recurso e uma fonte da definição do bean ajuda a determinar possíveis erros de configuração e fornece uma melhor experiência do desenvolvedor.

A partir da versão 5.4.3, o canal de erro padrão é configurado com a propriedade `requireSubscribers = true` para não ignorar silenciosamente as mensagens quando não há assinantes neste canal (por exemplo, quando o contexto do aplicativo é interrompido). Nesse caso, um `MessageDispatchingException` é lançado, o que pode emprestar no retorno de chamada do cliente do adaptador do canal de entrada para reconhecer negativamente (ou reverter) uma mensagem original no sistema de origem para reenvio ou outra consideração futura. Para restaurar o comportamento anterior (ignorar mensagens de erro não despachadas), a propriedade de integração global `spring.integration.channels.error.requireSubscribers` deve ser definida como `false`. Consulte Propriedades globais e `PublishSubscribeChannel` configuração (se você configurar um global `errorChannel` manualmente) para obter mais informações.

Versão 5.4.4
Última atualização 2021-02-17 19:24:37 UTC